Scheduling series-parallel task graphs to minimize peak memory

Enver Kayaaslan, Thomas Lambert, Loris Marchal, Bora Uçar

Scheduling series-parallel task graphs to minimize peak memory

Enver Kayaaslan[a], Thomas Lambert[b], Loris Marchal[c], Bora Uçar[c,*]

[a] *Inria and Université de Lyon, France*
[b] *ENS Lyon, France*
[c] *LIP, UMR5668 (CNRS - ENS Lyon - UCBL - Université de Lyon - INRIA), Lyon, France*

Abstract

We consider a variant of the well-known, NP-complete problem of minimum cut linear arrangement for directed acyclic graphs. In this variant, we are given a directed acyclic graph and we are asked to find a topological ordering such that the maximum number of cut edges at any point in this ordering is minimum. In our variant, the vertices and edges have weights, and the aim is to minimize the maximum weight of cut edges in addition to the weight of the last vertex before the cut. There is a known, polynomial time algorithm [Liu, SIAM J. Algebra. Discr., 1987] for the cases where the input graph is a rooted tree. We focus on the instances where the input graph is a directed series-parallel graph, and propose a polynomial time algorithm, thus expanding the class of graphs for which a polynomial time algorithm is known. Directed acyclic graphs are used to model scientific applications where the vertices correspond to the tasks of a given application and the edges represent the dependencies between the tasks. In such models, the problem we address reads as minimizing the peak memory requirement in an execution of the application. Our work, combined with Liu's work on rooted trees addresses this practical problem in two important classes of applications.

Keywords: series-parallel graphs, scheduling, peak memory minimization.

1. Introduction

A layout or a linear arrangement of a graph G is a total ordering of the vertices of G. In layout problems, the aim is to optimize a certain objective function. There are a number of very well-known layout problems which are surveyed by Díaz et al. [5, 21]. Among those problems CutWidth or the minimum cut linear arrangement (MCLA) is of immediate interest. Consider a

*Corresponding author. Tel: +33 472728932, 46 allée d'Italie, 69007, Lyon, France
Email addresses: enver.kayaaslan@ens-lyon.fr (Enver Kayaaslan), thomas.lambert@ens-lyon.fr (Thomas Lambert), loris.marchal@ens-lyon.fr (Loris Marchal), bora.ucar@ens-lyon.fr (Bora Uçar)

graph on n vertices and a layout. Consider a cut at i, where the first i vertices in the layout are on the left part and the remaining vertices are on the right part. The number of edges whose end points straddle the cut at i is called the width of the cut. The CutWidth problem asks for a layout in which the maximum width of a cut at positions 1 to $n-1$ is the minimum. This paper addresses a variant of the CutWidth problem in which the vertices and edges have weights, and the aim is to minimize the maximum value of a cut at position i for $i = 1, \ldots, n$, where the *value of a cut* at i is now defined as the sum of the weight of the vertex v at position i, of the weights of all edges whose one end point is either v or ordered before v and whose other end point is v or ordered after v. We address this problem for the well-known class of series-parallel graphs (a formal definition is given in Section 2 for completeness), and propose a polynomial time algorithm.

The CutWidth problem is NP-complete both in undirected graphs [10] and directed graphs [28], and solved efficiently for graphs with small treewidth [26] and small cutwidth [25]—see the survey and its addendum [5, 21] for a more thorough view of the known results. Leighton and Rao [15] discuss approximation algorithms for the CutWidth problem, which can be improved by using the heuristics by Arora et al. [1]. Yannakakis [29] presents a polynomial time algorithm for the CutWidth problem on trees (general trees) without weights; the weighted version remains NP-complete [19]. A historical source of interest in the CutWidth problem for graphs with unit weight edges (no vertex weights) is the pebble game of Sethi and Ullman [24], which is Pspace complete [11], and polynomial time solvable for rooted trees [23].

The variant of the CutWidth problem studied in this paper is first addressed by Liu [18] for the class of rooted trees. Liu discusses how the problem at hand corresponds to minimizing the peak memory in the context of a certain sparse direct solver [17]. For the sake of fidelity to Liu's original work, we use the term "peak memory" for defining the objective in our layout problem. Our main contribution in this paper is thus to expand the family of graph classes for which peak-memory problem can be solved in polynomial time.

Our results are of theoretical nature, yet we have been motivated by the practical problem of reducing the memory consumption of applications which are modeled as task graphs [22]. In these graphs, vertices represent tasks and edges represent the dependencies between tasks. Each task consumes one or more *input file/data*, and produces one or more *output file/data*. To be executed, a task also requires an *execution file/data*. As the size of the data to be processed increases, minimizing the peak memory of an application arises as an important objective, as the memory traffic is often the bottleneck. Consider the execution of a task graph on a single compute resource, i.e., a single processor and a single memory hierarchy. An execution of the application is defined by a traversal of the graph, that is, a schedule of the vertices which respects the dependencies (or topological order). The peak memory of a traversal is the maximum amount of memory needed to store application files/data at any given time throughout the application execution. The graphs with vertex and edges weights, and the definition of the value of a cut given above express the objec-

tive of minimizing the peak memory as a graph layout problem. Peak memory minimization problem has been addressed for applications whose task graphs are rooted trees [13, 14, 16, 18]. This work aspires to be helpful in scheduling applications whose task graphs are series parallel [3, 9, 20], as theoretical understanding of the underlying layout problem is needed to reduce the peak memory in a parallel execution environment (see for example a previous study [8]).

This paper is organized as follows. We present the problem formally in Section 2. We then describe the existing optimal algorithm for trees in Section 3, since this algorithm forms the basis of our proposed algorithms. In the same section, we present the principle of our algorithms on a subclass of SP-graphs. Then, Section 4 presents the proposed algorithm for general SP-graphs as well as the new notion on min-cut optimality needed to prove its correctness. Some of the arguments in our proofs are very lengthy and involved. For the sake of readability, the most demanding proofs are detailed in Appendix.

2. Peak memory minimization: Model and objective

We define the problem in general directed acyclic graphs (DAGs) with vertex and edge weights modeling applications. In this model, a DAG $G = (V, E, w_n, w_e)$ contains a vertex for each task of the application and a directed edge (p, q) between two vertices if the task corresponding to the vertex q needs a data produced by the task corresponding to the vertex p. Each task in the graph may have several input data, some execution data (or program), and several output data. We denote by $w_e(p, q) \geq 0$ the weight of edge (p, q) which represents the size of the data produced by task p for task q, and by $w_n(p) \geq 0$ the weight of a vertex p which is the size of the execution data of task p.

During the execution of a task (vertex) p, the memory must contain its input data, the execution data, and its output data. The memory needed for executing p is thus:

$$\left(\sum_{(r,p)\in E} w_e(r,p)\right) + w_n(p) + \left(\sum_{(p,q)\in E} w_e(p,q)\right) .$$

After p has been processed, its input and execution data are discarded, while its output data are kept in memory until they are consumed at the end of the execution of the corresponding tasks. For example, in the graph depicted in Figure 1a, if task A is processed first, 8 units of memory are needed for its processing, but only 4 remains in the memory after its completion. If task C is processed right after task A, 7 units of memory are needed during its processing (4 to store the data produced by A for B, and 3 for the execution and output data of C).

While processing the task graph, memory consumption changes as data are created and deleted. Our objective is to minimize the *peak memory*, i.e., the maximum amount of memory used in the graph traversal.

More formally, we define a schedule of the graph as a total order (layout) π on the vertices, denoted with $\leq_\pi$, such that $p \leq_\pi q$ means that vertex p is

ordered before vertex q. The precedence constraints of the task graphs impose that this is a topological order: we have $p \leq_\pi q$ for any edge $(p, q) \in E$. We use the notation $\max^\pi$ (resp. $\min^\pi$) to express the maximum (resp. minimum) according to the order π: $\min^\pi G$ is, for instance, the first vertex scheduled by π. When considering a subset X of vertices, we denote by $\pi[X]$ the order π restricted to this subset. A schedule is also represented as a list of vertices: $\pi = \langle 1, 2, 3\rangle$ denotes that $1 \leq_\pi 2 \leq_\pi 3$.

We define $\mu(p, \pi)$ as the memory required during the execution of a vertex $p \in V$ under the schedule π as

$$\mu(p, \pi) = w_n(p) + \sum_{q \leq_\pi p \leq_\pi r} \{w_e(q, r) : (q, r) \in E\} .$$

Note that the edges (q, r) such that $q \leq_\pi p \leq_\pi r$ correspond to the data that have been created but not yet consumed while p is being processed. The objective is to find a schedule π for a graph G that minimizes the *peak memory*, defined as

$$\mu(\pi) = \max_{p \in V} \mu(p, \pi) .$$

Given a directed graph $G = (V, E, w_n, w_e)$, we define the *reverse graph* $\bar{G} = (V, \bar{E}, w_n, w_e)$ where the orientation of all edges is changed: $\bar{E} = \{(j, i),\ (i, j) \in E\}$. Note that vertex and edge weights are kept unchanged. Given a schedule π of G, we may build the reverse schedule $\bar{\pi}$ such that $q \leq_{\bar{\pi}} p$ whenever $p \leq_\pi q$. It is straightforward to check that both schedules have the same peak memory on their respective graph: $\mu_G(\pi) = \mu_{\bar{G}}(\bar{\pi})$.

In this paper, we concentrate on series-parallel graphs, which are defined as follows (see for example [7] for more information).

Definition 1. *A two-terminal series-parallel graph, or SP-graph, G with terminals s and t is recursively defined to be either:*

Base case: *A graph with two vertices s and t, and an edge (s, t).*

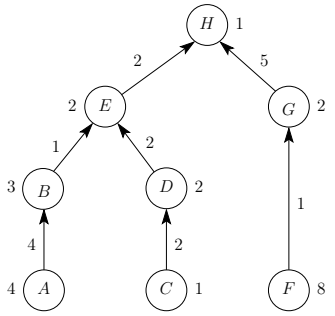
Series composition: *The* series composition *of two SP-graphs G_1 with terminals s_1, t_1 and G_2 with terminals s_2, t_2 formed by identifying $s = s_1$, $t = t_2$ and $t_1 = s_2$, and denoted by $\langle G_1, G_2\rangle$;*

Parallel composition: *The* parallel composition *of two SP-graphs G_1 with terminals s_1, t_1 and G_2 with terminals s_2, t_2 formed by identifying $s = s_1 = s_2$ and $t = t_1 = t_2$, and denoted by $\{G_1, G_2\}$.*

The vertices s and t are called source *and* target *of the graph.*

When the graph is defined this way, the dependencies are from the source vertices to the target vertices. Series-parallel graphs can be recognized and decomposed into a tree of series and parallel combinations in linear time [27].

While solving the peak memory minimization problem for SP-graphs, we will need a solution for a sub-family of SP-graphs, which are called *parallel-chains* and are defined below. A sample parallel-chain graph is shown in Fig. 2.



(a) Tree with vertex and edge weights.

task being processed	A		B		C		D		E	
memory	8	4	8	1	4	3	7	3	7	2

(b) Memory profile for the traversal (A, B, C, D, E)

segments	(hill,valley) values
(A, B)	(8,1)
(C, D, E)	(7,2)

(c) Segments for the traversal (A, B, C, D, E)

segments	(hill,valley) values
(F)	(9,1)
(G)	(8,5)

(d) Segments for the traversal (F, G)

Figure 1: Sample in-tree and decomposition of its schedule in segments.

Definition 2. *A* chain *is a two-terminal series-parallel graph obtained without using any parallel composition. A* parallel-chain graph *is a two-terminal series-parallel graph obtained by the unique parallel composition of a number of chains.*

3. Solving the peak memory problem on trees and parallel-chain graphs

We first recall Liu's algorithm [18] for solving the peak memory minimization problem on rooted trees. We then propose its adaptation for parallel-chain graphs; this new algorithm is used later (in Section 4) as a building block for solving the peak memory problem on series-parallel graphs. The algorithm proposed by Liu applies to *in-trees*, that is trees whose dependencies are directed towards the root (contrarily to *out-trees*, where dependencies are directed towards leaves). Note that if T is an in-tree and π is the schedule of T computed by Liu's algorithm, then $\bar{\pi}$ is also a peak memory minimizing schedule of the out-tree $\bar{T}$.

3.1. Liu's algorithm for trees

Liu [18] proposes an algorithm to find an optimal tree traversal for peak memory minimization. Liu's original work concerns a different model where there is no edge weights (there are only vertex weights). We present here its immediate adaptation to our model.

Liu's algorithm is shown in Algorithm 1. To compute an optimal traversal for a tree rooted at vertex k, it recursively computes an optimal traversal for the subtrees rooted at the children of k, then merges theses optimal traversals, and finally appends k to the end. Merging the traversals of the children is the sophisticated part of the algorithm. There is no reason to schedule the subtrees one after the other; an optimal schedule may switch from one subtree to another. Liu makes the observation that in an optimal traversal the switching points between the subtrees' processing have to be local minima in the memory

Algorithm 1 LIU-TREE-SCHEDULE(T)

Require: $T = (V, E, w_n, w_e)$: tree with vertex- and edge-weights.
Ensure: π: Schedule with the minimum peak memory $\mu(\pi)$

▶ Base case
if $V = \{u\}$ **then**
  return $\langle u \rangle$

▶ General case
Let r be the root of T and $T_1, T_2, \ldots, T_k$ its subtrees
for $i = 1$ to k **do**
  $\pi_i \leftarrow$ LIU-TREE-SCHEDULE(T_i)
  Compute the hill-valleys segments $s_1^i, \ldots, s_k^i$ of T_i in schedule π_i as in Definition 3
Sort all segments of all subtrees in non-increasing $(hill - valley)$ value
Based on this segment ordering, order the vertices in each segment consecutively, followed by the root r, to build π. Within each segment s_i^j, vertices are ordered according to π_i.

profile: while processing one subtree T_i, there is no reason to switch to T_j if one can reduce the memory needed for T_i by processing one more task in T_i. This leads to slicing the traversal into atomic parts, called *segments*. The end-points of segments (which are some particular local minima in the memory profile) are called *valleys*, while the peak memory vertices of each segment are called *hills*. The segments and their hill/valley values are formally defined as follows.

Definition 3. *Let G be an in-trees and π be a traversal of G. The first segments of π consists in nodes $u \leq_\pi v_1$ and the ith segments of π (for $i \geq 2$) contains nodes u such that $v_{i-1} <_\pi u \leq_\pi v_i$ with:*

- *M_1^h is the peak memory of the whole traversal;*
- *M_i^v is the minimum amount of memory occurring after the step when M_i^h is (last) attained, for $i \geq 1$;*
- *M_i^v is (last) attained on v_i, for $i \geq 1$;*
- *M_i^h (for $i \geq 2$) is the peak memory after v_i.*

The sequence of hill-valley ends when the last vertex is reached. The segments consist of the vertices comprised between two valleys. For example, for the tree depicted in Figure 1a, consider the traversal (A, B, C, D, E) of the subtree rooted in E. The memory occupation during and after the processing of the tasks and the segments that are deduced from this memory profile are illustrated in Figure 1c. Similarly, the segments of the traversal (F, G) of the subtree rooted in G are detailed on Figure 1d.

To merge the traversals of the subtrees, the first step is to compute all hill-valley segments. Then the lists of segments are merged using the following criterion: if several segments are available (one for each subtree in the beginning), it is always beneficial to start with the segment with the maximum $(hill - valley)$

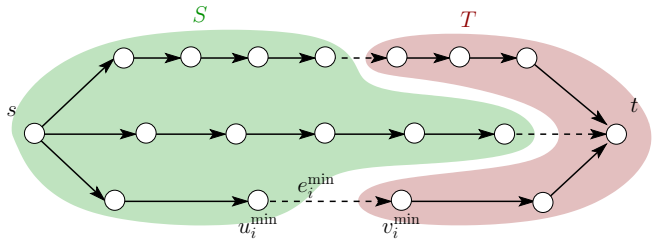


Figure 2: Sample parallel-chain graph. This graph's decomposition is used for Lemma 1. If the dashed edges are minimum in the corresponding chains, an optimal traversal can be found by first ordering the vertices in the set S and then the vertices in the set T.

difference. Intuitively, the residual memory will only increase when processing segments, so it seems better to start with (i) segments with larger peak memory (hill) to avoid large memory consumption later, and (ii) segments with smaller residual memory (valley) to ensure an increase of memory as small as possible. The tradeoff between both criteria is obtained using the difference $(hill - valley)$.

In the example of Figure 1, when merging the traversals from the subtrees rooted at E and G at the vertex H, we start by comparing the first segments of each subtree: segment (F) is selected because it has a larger $(hill - valley)$ value of $9 - 1 = 8$. This segment is ordered, and then removed from the segments list, and we proceed by comparing segments (A, B) and (G): (A, B) has larger $(hill - valley)$ value of $8 - 1 = 7$, so it is selected. By iterating this process, we end up with the following ordering: $(F), (A, B), (C, D, E), (G)$. The last step is to add the root H of the subtree to the traversal.

Liu proves that Algorithm 1 is correct and its worst-case runtime complexity is $O(n^2)$. We note that even if this algorithm is designed for in-trees, we can compute an optimal schedule of an out-tree by reversing all edges to obtain an in-tree, applying this algorithm, and then reversing the obtained schedule.

3.2. Algorithm for parallel-chain graphs

The main idea (summarized in Fig. 2) is to remove one edge from each chain, so as to disconnect the graph into one out-tree (the part S in the figure) and one in-tree (the part T in the figure). Then, we can reuse Liu's algorithm to compute an optimal traversal for these two trees. The following lemma states that if the removed edges are of minimal weight in each chain, it is possible to first schedule all the vertices that are *before* this minimal cut, and then all the vertices *after* the cut, without increasing the peak memory.

Lemma 1. *Let G be a parallel-chain graph. For each chain C_i of this graph, let $e_i^{\min} = (u_i^{\min}, v_i^{\min})$ be an edge of C_i with the minimum weight. Let S be the set of ancestors of the $u_i^{\min}$'s, including them. Let T be the set of successors of the $v_i^{\min}$'s, including them. Let π be a schedule of G and γ be the schedule obtained from π by scheduling all vertices of S before vertices of T, formally $\gamma = \langle \pi[S], \pi[T] \rangle$. Then, $\mu(\gamma) \leq \mu(\pi)$.*

This intuitive result can be proved by using Theorem 2, Corollary 1 and Theorem 1 given below. We do not formalize the proof, since the objective of this section is to give an intuition on the algorithm for general SP-graphs.

Thanks to this result, we know that there exists an optimal schedule which ordered first vertices from S, and then vertices from T. Assume for a moment that the weight of all $e_i^{\min}$ edges is zero ($w_e(e_i^{\min}) = 0$). Then, it is as if the graph was disconnected, and we have two separate trees to schedule. T is an in-tree, and Liu's algorithm can compute an optimal schedule π for it. S is an out-tree, so that $\bar{S}$ is an in-tree: if γ is the optimal schedule computed by Liu's algorithm for $\bar{S}$, $\bar{\gamma}$ is optimal for S. Then, $\langle \bar{\gamma}, \pi \rangle$ is an optimal schedule of the whole graph. This approach can be generalized to parallel-chain graphs with non-zero weights on the minimal edges, as stated in Algorithm 2. For such graph, the weight of the minimal edges is subtracted from all edges of a chain, and is added to the weight of all vertex of the chain (except the terminals). By setting $C = \sum_i w_e(e_i^{\min})$, it is easy to verify that for any schedule π, the memory footprint during the execution of a node or after its execution in the modified graph is the same memory as in the original graph minus C. Thus, any optimal schedule for the modified graph is an optimal schedule for the original graph. We will prove that this algorithm computes an optimal schedule of all parallel-chain graphs (Theorem 8).

Algorithm 2 PARALLEL-CHAINS-SCHEDULE(T)

Require: $PC = (V, E, w_n, w_e)$: parallel-chain graph with vertex- and edge-weights.
Ensure: π: Schedule with minimal peak memory $\mu(\pi)$
  Let $C_1, \ldots, C_q$ be the chains of PC
  for $i = 1$ to q **do**
    Let $e_i^{\min} = (u_i^{\min}, v_i^{\min})$ be the edge of minimum weight in C_i
    Remove the edge $e_i^{\min}$ from the graph
    Update the weight of each other edge e of C_i: $w_e(e) \leftarrow w_e(e) - w_e(e_i^{\min})$
    Update the weight of each vertex u in C_i (except s and t): $w_n(u) \leftarrow w_n(u) + w_e(e_i^{\min})$
  Consider the two trees T^{out}, T^{in} obtained after the removal of $e_1^{\min}, \ldots,$
  Let $\bar{T}^{out}$ be the in-tree obtained by reversing all edges in T^{out}
  $\pi_1 \leftarrow$ LIU-TREE-SCHEDULE($\bar{T}^{out}$)
  $\pi_2 \leftarrow$ LIU-TREE-SCHEDULE(T^{in})
  return $\langle \overline{\pi_1}, \pi_2 \rangle$

4. Solving the peak memory problem for series-parallel graphs

This section contains the proposed polynomial time algorithm to compute a peak memory minimizing schedule of series-parallel graphs. Let us first give a verbal overview of the final algorithm. As is common, the algorithm relies on the recursive structure of the series-parallel graphs. To solve the peak memory

problem for a series-parallel graph G which is a composition of G_1 and G_2, we first recursively solve the peak memory problem on G_1 and G_2. If G is a series composition of G_1 and G_2, it is straightforward to obtain a schedule for G by concatenating those for G_1 and G_2. If G is a parallel composition of G_1 and G_2, we first create a chain (according to the optimal schedules found) for G_1 and G_2. By identifying the terminal vertices in G_1 and G_2, we obtain a parallel-chain graph on which the peak memory problem is solved using the algorithm proposed in Section 3.2. This results in a peak memory minimizing schedule for the initial graph G. The algorithm is simple and intuitive as it follows the recursive definition of series-parallel graphs. However, the the proof of optimality of the algorithm is complex and involved. This is not surprising as it was also the case for Liu's algorithm on trees [18], which extended an already involved algorithm [29].

The formal algorithm (Section 4.4) is an adaptation of Liu's algorithm for vertex weighted trees. In order to facilitate this adaptation, we define a new graph model (presented in Section 4.1) in which we have only vertex weights. This model is simpler, yet expressive enough to capture the peak memory objective. We then define a new relation on schedules and a new objective function (Sections 4.2 and 4.3) in this modem that are needed to obtain the proof of its optimality (which is presented in Section 4.5). Some of the detailed proofs are delegated to the Appendix.

4.1. A simpler model with only vertex weights

We introduce here a new graph model, called the *cumulative weight* model, which has only vertex weights (no edge weights). We prove that this model can emulate the peak-memory minimization problem. Formally, let $G = (V, E, \omega)$, where $\omega : V \to \mathbb{Z}$ is a weight function on the vertices. For a given set $U \subseteq V$ of vertices, $\omega(U)$ denotes the sum of weights of vertices in U, i.e., $\omega(U) = \sum_{v \in U} \omega(v)$. The vertex-weight function ω should satisfy $\omega(G) = 0$. Let π be a schedule and for each vertex $v \in V$, consider the cumulative sum of the weighs of all vertices scheduled before v:

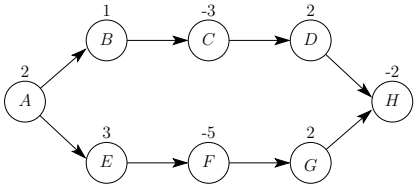
$$\Sigma(v, \pi) = \omega(\{u \in V,\ u \leq_\pi v\}) \ . \tag{1}$$

We define the *cutwidth* $\rho(\pi)$ of a schedule π as the maximum of all these sums:

$$\rho(\pi) = \max_{p \in V} \omega(\{q \in V,\ q \leq_\pi p\}) = \max_{p \in V} \Sigma(p, \pi) \ . \tag{2}$$

Figure 3 presents a simple SP graph in the cumulative weight model as well as three schedules π with cutwidth 3 (which is minimal), and γ and λ with cutwidth 6.

Note that the cutwidth as defined in the cumulative weight model is a straightforward adaptation of the CutWidth problem (or minimum cut linear arrangement) as presented in the introduction: from a graph with only edge weights w_e, for each edge (i, j) we simply set $\omega(i) = w_e(i, j)$ and $\omega(j) = -w_e(i, j)$ to obtain the same problem with the cumulative weight model.



(a) Simple SP graph in the cumulative weight model. Note that $\omega(G) = 0$ as imposed in the model.

p	A	B	C	E	F	D	G	H
$\Sigma(p, \pi)$	2	3	0	3	-2	0	2	0

(b) Schedule π

p	A	B	E	C	F	D	G	H
$\Sigma(p, \gamma)$	2	3	6	3	-2	0	2	0

(c) Schedule γ

p	A	E	B	C	F	D	G	H
$\Sigma(p, \lambda)$	2	5	6	3	-2	0	2	0

(d) Schedule λ

Figure 3: Simple SP-graph in the cumulative weight model and two possible schedules.

Thus, the algorithm presented below for SP-graphs in the cumulative weight model also solves the classical CutWidth problem on SP-graphs with edge weights. We now exhibit a similar reduction from the peak memory problem presented in Section 2.

4.1.1. Minimizing the cutwidth allows to minimize the peak memory

Below (Theorem 1), we show that by minimizing the cutwidth in the cumulative weight model, one can minimize the peak memory in the original model. For this purpose we need some definitions. Given an instance $G = (V, E, w_n, w_e)$ of the peak memory problem, we construct an instance of the cumulative weight problem, with a (directed) bipartite graph $G_B = (V_B, E_B, \omega)$ as follows. For each vertex $p \in V$, we introduce a pair of vertices $p_{start}, p_{stop} \in V_B$ such that $(p_{start}, p_{stop}) \in E_B$. The vertex p_{start} represents the beginning of the computation corresponding to the task p, while p_{stop} represents the end of this computation. For each edge $(p, q) \in E$, we add an edge $(p_{stop}, q_{start}) \in E_B$. For each vertex p, we set

$$\omega(p_{start}) = w_n(p) + \sum_{(p,r)\in E} w_e(p, r) \ , \tag{3}$$

to represent the allocation of temporary and output data in the memory at the beginning of the task p, and

$$\omega(p_{stop}) = -w_n(p) - \sum_{(q,p)\in E} w_e(q, p) \ , \tag{4}$$

to represent the deallocation of temporary and input data from the memory at the end of the task p. Note that with these definitions $\omega(G_B) = 0$. Let π_B be a schedule of G_B. It is easy to see that a schedule can be constructed for G using the order of "stop" vertices in π_B. Given π_B, let π be a schedule of G such that $p \leq_\pi q$ whenever $p_{stop} \leq_{\pi_B} q_{stop}$. Observe that $\sum(v_{stop}, \pi_B) = \sum_{x \leq_\pi v <_\pi y} w_e(x, y)$, that is the cumulative weight of v_{stop} corresponds to the weight of the edges (in G) which are cut by an imaginary line just after v under

the schedule π. We now show how minimizing the cutwidth in G_B minimizes the peak memory in G.

Theorem 1. *Let π_B be a schedule of G_B whose cutwidth $\rho(\pi_B)$ is minimum for G_B and π be the corresponding schedule of G. Then, $\mu(\pi)$ is the minimum peak memory for G.*

PROOF — We provide the outline of the proof by using lemmas proved in Appendix A. By Lemma A.4, we can assume without loss of generality that p_{start} and p_{stop} are consecutive for all p in π_B. This implies that $\mu(\pi) = \rho(\pi_B)$ by Lemma A.3. Suppose for the sake of contradiction that $\mu(\pi)$ is not minimum. In other words, there is a schedule γ of G where $\mu(\gamma) < \mu(\pi)$. Then, we can construct γ_B for G_B from γ by replacing each vertex p with p_{start} and p_{stop}. By Lemma A.3 again, $\mu(\gamma) = \rho(\gamma_B)$ and hence $\rho(\gamma_B) < \rho(\pi_B)$, a contradiction. □

4.1.2. Reverse graph in the cumulative weight model

We now show a few useful results in the proposed cumulative weight model that uses the reverse graph. Remember that in our final algorithm, we will disconnect a parallel composition into both an in-tree and an out-tree, and that we will apply Liu's algorithm on the reverse of the in-tree. Thus, it is important that to determine the cutwidth of the reverse graph, as done in the following two lemmas.

We first recall and extend the notion of reverse graph from Section 2 to the vertex-weighted model. Given a graph $G = (V, E, \omega)$, its reverse graph $\bar{G} = (V, \bar{E}, \bar{\omega})$ has the same set of vertices as G, however all edges are reversed $\bar{E} = \{(q, p) : (p, q) \in E\}$ and all vertex weights take the opposite sign of their value in G: $\bar{\omega}(p) = -\omega(p)$ for any $p \in V$. The reverse of a schedule π is defined as in Section 2: $q \leq_{\bar{\pi}} p$ whenever $p \leq_{\pi} q$, for any pair of vertices p, q.

The first result links the cumulative sums up to some vertex p in both π and $\bar{\pi}$.

Lemma 2. *For any vertex $p \in G$, we have*

$$\Sigma(p, \pi) - \Sigma(p, \bar{\pi}) = \omega(p), \tag{5}$$

where $\Sigma(p, \pi)$ and $\Sigma(p, \bar{\pi})$ are defined over G and $\bar{G}$, respectively.

PROOF — Notice that $\{q \leq_{\bar{\pi}} p\} = \{q \geq_{\pi} p\}$ due to the definition above. Recall that $\omega(G) = 0$, and thus $\bar{\omega}(\bar{G}) = -\omega(G) = 0$. Then,

$$\begin{aligned}
\Sigma(p, \bar{\pi}) &= \bar{\omega}(\{q \leq_{\bar{\pi}} p\}) = \bar{\omega}(\{q \geq_{\pi} p\}) \\
&= \bar{\omega}(\bar{G}) - \bar{\omega}(\{q <_{\pi} p\}) \\
&= -\bar{\omega}(\{q <_{\pi} p\}) \\
&= \omega(\{q <_{\pi} p\}) = \Sigma(p, \pi) - \omega(p).
\end{aligned}$$

□

The second result shows that a schedule π has the same cutwidth as its reverse $\bar{\pi}$.

Lemma 3. *For any schedule π of G, $\rho(\pi) = \rho(\bar{\pi})$.*

PROOF — Let $\bar{p}^*$ be a vertex such that $\rho(\bar{\pi}) = \Sigma(\bar{p}^*, \bar{\pi})$ and q be its predecessor under π. Then,

$$\begin{aligned}\rho(\pi) \geq \Sigma(q,\pi) &= \Sigma(\bar{p}^*,\pi) - \omega(\bar{p}^*) && \text{holds by the definitions (1) and (1)}\\ &= \Sigma(\bar{p}^*,\bar{\pi}) && \text{by Equation (5)}\\ &= \rho(\bar{\pi}).\end{aligned}$$

By noting that the roles of π and $\bar{\pi}$ can be changed in the above lines, we obtain $\rho(\bar{\pi}) \geq \rho(\pi)$ and hence the equality $\rho(\pi) = \rho(\bar{\pi})$. □

4.2. A new relation on schedules

We present here a relation on schedules, denoted by $\preceq$. In order to prove the optimality of our algorithm, we first prove that it produces a schedule which is minimal for $\preceq$, and that this implies cutwidth optimality. The $\preceq$ relation is stronger than cutwidth comparison and includes all the assumptions needed for the induction. This reasoning is largely implies by Liu's proof for trees [18], and the new relation itself is the translation of the partial orders on subtree schedules that he introduced.

4.2.1. Definition of the relation on schedules

Given a directed graph G, two vertices p and q of G, and two schedules π and γ of G, we say that the schedule π at p *dominates* the schedule γ at q (and we write $p \ {}_{\pi}\!\!\rightarrow_{\gamma} q$) if and only if

C1. $\Sigma(p,\pi) \leq \Sigma(q,\gamma)$, and

C2. $\min\{\Sigma(s,\pi) : s \geq_{\pi} p\} \leq \min\{\Sigma(r,\gamma) : r \geq_{\gamma} q\}$.

Note that the second condition (C2) is equivalent to stating that for any $r \geq_{\gamma} q$, there exists $s \geq_{\pi} p$ such that $\Sigma(s,\pi) \leq \Sigma(r,\gamma)$.

Now, we define the relation $\preceq$ on the set of schedules for G as follows.

Definition 4 (Relation $\preceq$). *Let π and γ be two schedules of G. We say that $\pi \preceq \gamma$ for G, if for each $p \in G$, there is a vertex $q \in G$ such that $p \ {}_{\pi}\!\!\rightarrow_{\gamma} q$.*

In the sample graph presented in Fig. 3, we have $\pi \preceq \gamma$ and $\pi \preceq \lambda$ since for each vertex p, $p \ {}_{\pi}\!\!\rightarrow_{\gamma} E$ and $p \ {}_{\pi}\!\!\rightarrow_{\lambda} E$. We also have $\gamma \preceq \lambda$ and $\lambda \preceq \gamma$ since

$$\begin{aligned}\text{for } p \in \{A,B,C,E,F\}, &\quad p \ {}_{\gamma}\!\!\rightarrow_{\lambda} B \text{ and } p \ {}_{\lambda}\!\!\rightarrow_{\gamma} E,\\ \text{for } p \in \{D,G,H\}, &\quad p \ {}_{\gamma}\!\!\rightarrow_{\lambda} G \text{ and } p \ {}_{\lambda}\!\!\rightarrow_{\gamma} G.\end{aligned}$$

This example illustrates that the $\preceq$ relation is not anti-symmetric, therefore it is not a partial order (in Liu's case [18], the relation is partial order). We prove that $\preceq$ is reflexive and transitive, and thus is a preorder.

Lemma 4. *The relation $\preceq$ is a preorder (reflexive and transitive).*

PROOF — We need to show that the relation $\preceq$ is both reflexive and transitive. Let α, β, and θ be schedules of G.

(i) *Reflexivity.* Since $p \ {}_{\alpha}\!\!\rightarrow_{\alpha} p$ for each $p \in V$, $\alpha \preceq \alpha$, and thus, $\preceq$ is reflexive.

(ii) *Transitivity.* Assume that $\alpha \preceq \beta$ and $\beta \preceq \theta$. Take any $p \in G$. Let v and q be such that $p \ {}_{\alpha}\!\!\rightarrow_{\beta} v$ and $v \ {}_{\beta}\!\!\rightarrow_{\theta} q$. We claim that $p \ {}_{\alpha}\!\!\rightarrow_{\theta} q$, as well. There are two conditions we should examine.

1) (C1). This follows as $\Sigma(p,\alpha) \leq \Sigma(v,\beta)$ and $\Sigma(v,\beta) \leq \Sigma(q,\theta)$, due to the first conditions of $p \ {}_{\alpha}\!\!\rightarrow_{\beta} v$ and $v \ {}_{\beta}\!\!\rightarrow_{\theta} q$.

2) (C2). Take any vertex $r \geq_{\theta} q$. Then, there exists $s \geq_{\alpha} p$ and $w \geq_{\beta} v$ such that $\Sigma(s,\alpha) \leq \Sigma(w,\beta) \leq \Sigma(r,\theta)$, as a consequence of the second conditions of $p \ {}_{\alpha}\!\!\rightarrow_{\beta} v$ and $v \ {}_{\beta}\!\!\rightarrow_{\theta} q$.

□

4.2.2. Minimality for the $\preceq$ relation implies cutwidth optimality

In the example of Fig. 3, we noticed that $\pi \preceq \gamma$ and $\pi \preceq \lambda$ while π has a smaller cutwidth than both γ and λ. This is actually a major property of the $\preceq$ relation: it implies a comparison on the cutwidth, not only for the schedules being compared but also for their reverse schedule, as expressed by the following lemma and its corollary.

Lemma 5. *If $\pi \preceq \gamma$ then $\rho(\pi) \leq \rho(\gamma)$ and $\rho(\bar{\pi}) \leq \rho(\bar{\gamma})$.*

PROOF — Assume that $\pi \preceq \gamma$. Then for any $p \in V$, there exists $q \in V$ such that $p \ {}_{\pi}\!\!\rightarrow_{\gamma} q$, and thus, $\Sigma(p,\pi) \leq \Sigma(q,\gamma)$. Then, by definition, $\rho(\pi) \leq \rho(\gamma)$. Moreover, thanks to Lemma 3, we have $\rho(\bar{\gamma}) = \rho(\gamma) \geq \rho(\pi) = \rho(\bar{\pi})$. □

Corollary 1. *If π is minimal for $\preceq$ on G, then π and $\bar{\pi}$ have the minimum cutwidths (on G and $\bar{G}$ respectively).*

4.3. Min-cut optimality

We are now ready to present the central notion needed to provide an optimal algorithm for series-parallel graphs, namely the min-cut optimality. We show that a min-cut optimal schedule is also cutwidth optimal.

4.3.1. Definition of min-cut optimality

Min-cut optimality is based on the classical notion of *cut* in a graph: a cut (S, T) of G is defined as a bisection of its vertices into two nonempty sets S and T. We say that a cut is *topological* if there exists no edge $(p, q) \in E$ such that $p \in T$ and $q \in S$ (or equivalently, for any edge $(p, q) \in E$, we have either $p \in S$ or $q \in T$). The *width* of a topological cut (S, T) is defined as $c(S, T) = \omega(S)$.

Consider a topological cut (S, T) and a schedule π. If in π all vertices of S appear before all vertices of T, that is, for $p \in S$ and $q \in T$, we have $p \leq_\pi q$, then we say that π is in *compliance* with (S, T). In this case, we have

$$c(S, T) = \Sigma(p^*, \pi), \text{ where } p^* = \max^\pi S. \tag{6}$$

We say that a topological cut (S, T) is *minimum*, or a min-ω-cut, if its width is minimum, that is,

$$c(S, T) = \min\{c(S', T') \text{ where } (S', T') \text{ is a topological cut}\}.$$

We define by $G[S]$ the subgraph of G which contains only the vertices in S and the edges between these vertices. Similarly, for any schedule π of G, $\pi[S]$ is the schedule induced by π on $G[S]$.

Definition 5 (Min-cut optimality). *Let (S, T) be a minimum topological cut. A schedule π is* cut-optimal *with (S, T), if π is in compliance with (S, T), $\bar{\pi}[S]$ is minimal for $\preceq$ on $\bar{G}[S]$, and $\pi[T]$ is minimal for $\preceq$ on $G[T]$. Furthermore, π is called* min-cut optimal *if π is cut-optimal with some min-ω-cut of G.*

We now state a simple lemma which will be useful in the following.

Lemma 6. *Let π be a schedule of G. Let (S, T) be any minimum topological cut of G. Then,*

$$c(S, T) \leq \Sigma(p, \pi),$$

for any $p <_\pi \max^\pi G$.

PROOF — Take any $p <_\pi \max^\pi G$. Then suppose for the sake of contradiction that $\Sigma(p, \pi) < c(S, T)$. Consider the topological cut (S^p, T^p), where $S^p = \{v \leq_\pi p\}$. Then, by (6), we have $c(S^p, T^p) = \omega(S^p) = \Sigma(p, \pi) < c(S, T)$, which contradicts that $c(S, T)$ is minimum. □

4.3.2. Properties of min-cut optimality

We present two important properties of min-ω-cuts. The first property, shown in the next theorem, is that transforming a schedule so that it is in compliance with a min-ω-cut will not make it larger in the sense of the $\preceq$ relation (and thus, thanks to Lemma 5, will not increase its cutwidth). It generalizes Lemma 1, which was limited to parallel-chain graphs and expressed in the original graph model.

Theorem 2. *Let G be an acyclic graph with a single source vertex and a single sink vertex, and γ be a schedule of G. Let (S,T) be a min-ω-cut of G, and $\pi = \langle \gamma[S], \gamma[T] \rangle$. Then, $\pi \preceq \gamma$ and $\bar{\pi} \preceq \bar{\gamma}$.*

PROOF — Since (S,T) is a topological cut of G, π is a schedule of G. Note that γ may not be in compliance with (S,T).

The proof relies on the two following properties: for each $p \in G$, $p \ {}_{\pi}\!\!\rightarrow_{\gamma} p$ (which proves that $\pi \preceq \gamma$ according to the definition of this relation) and $p \ {}_{\bar{\pi}}\!\!\rightarrow_{\bar{\gamma}} p$ (which proves $\bar{\pi} \preceq \bar{\gamma}$). We only provide the proof of the first property, as the second one can be proved using the very same arguments.

Let s^+ be the last scheduled vertex of S in γ. Similarly, let t^- be the first scheduled vertex of T in γ. Formally,

$$s^+ = \max^{\gamma} S, \quad t^- = \min^{\gamma} T.$$

Note that s^+ is also the last scheduled vertex of S in π (and t^- the first scheduled vertex of T in π) and $\Sigma(s^+, \pi) = c(S,T)$.

To prove that $p \ {}_{\pi}\!\!\rightarrow_{\gamma} p$ for $p \in G$, we investigate $\Sigma(p,\pi)$ and $\Sigma(p,\gamma)$. First, for any $p \in V$, we consider s_p (respectively t_p), the last scheduled vertex of S (resp. of T) that does not come after p in γ:

$$s_p = \max^{\gamma}\{s \in S,\ s \leq_{\gamma} p\}, \quad t_p = \max^{\gamma}\{t \in T,\ t \leq_{\gamma} p\}.$$

Note that s_p is always defined since G has a single source vertex. However, t_p may not be defined, as the set $\{t \in T,\ t \leq_{\gamma} p\}$ may be empty, which occurs when $p \in S$ and p comes before t^- in γ; in this case $p = s_p$. If t_p is defined, we have:

$$\begin{aligned} \Sigma(p,\gamma) &= \Sigma(s_p, \gamma[S]) + \Sigma(t_p, \gamma[T]) \\ &= \Sigma(s_p, \pi[S]) + \Sigma(t_p, \pi[T]) \\ &= \Sigma(s_p, \pi) + \Sigma(t_p, \pi) - \Sigma(s^+, \pi) \\ &= \Sigma(s_p, \pi) + \Sigma(t_p, \pi) - c(S,T). \end{aligned} \tag{7}$$

As a result,

$$\Sigma(p,\gamma) = \begin{cases} \Sigma(p,\pi) & \text{if } t_p \text{ is not defined,} \\ \Sigma(t_p,\pi) + \Sigma(s_p,\pi) - c(S,T) & \text{otherwise.} \end{cases} \tag{8}$$

We are now ready to show $p \ {}_{\pi}\!\!\rightarrow_{\gamma} p$, for any $p \in V$. We verify the two conditions defining this statement given in Section 4.2.1 for any $p \in V$, as follows.

1. The first condition to verify is C1: $\Sigma(p,\pi) \leq \Sigma(p,\gamma)$. When t_p is not defined, this condition directly derives from Equation (8). Consider now that t_p is defined. We consider two cases depending on which side of the cut p lies:

(a) ($p \in S$). Then, $s_p = p$. This implies

$$\Sigma(p, \gamma) = \Sigma(p, \pi) + \Sigma(t_p, \pi) - c(S, T).$$

Note that $t_p <_\gamma p <_\gamma \max^\gamma G = \max^\pi G$, since $p \in S$. Since $\Sigma(t_p, \pi) \geq c(S, T)$ by Lemma 6 applied on π, the condition C1 is met.

(b) ($p \in T$). Then, $t_p = p$. This implies

$$\Sigma(p, \gamma) = \Sigma(p, \pi) + \Sigma(s_p, \pi) - c(S, T).$$

Note that $s_p <_\gamma \max^\gamma G = \max^\pi G$, since $s_p \in S$. Since $\Sigma(s_p, \pi) \geq c(S, T)$ by Lemma 6, the condition C1 is met.

2. The second condition to verify (C2) is

$$\min\{\Sigma(s, \pi) : s \geq_\pi p\} \leq \min\{\Sigma(r, \gamma) : r \geq_\gamma p\}$$

(or equivalently, for any $r \geq_\gamma p$, there exists $s \geq_\pi p$ such that $\Sigma(s, \pi) \leq \Sigma(r, \gamma)$). First, note that the last vertex is the same in both schedules: $r_{end} = \max^\pi G = \max^\gamma G$ and thus $\Sigma(r_{end}, \pi) = \Sigma(r_{end}, \gamma)$. Now, consider any $r \geq_\gamma p$ such that $r <_\gamma \max^\gamma G$. We consider the two following cases:

a) p is in S. Then, $\Sigma(s^+, \pi) = c(S, T) \leq \Sigma(r, \gamma)$ and $s^+ \geq_\pi p$.

b) p is in T. Let $t_r = \max^\gamma\{t \in T,\ t \leq_\gamma r\}$. Note that t_r is well defined, since $r \geq_\gamma p$. We also have $t_r \geq_\gamma p$. Then, by Equation (7),

$$\Sigma(r, \gamma) = \Sigma(s_r, \pi) + \Sigma(t_r, \pi) - c(S, T).$$

Since $\Sigma(s_r, \pi) \geq c(S, T)$ by Lemma 6, we have $\Sigma(t_r, \pi) \leq \Sigma(r, \gamma)$. Notice that $t_r \geq_\pi p$ since $t_r \geq_\gamma p$, and, both $p \in T$ and $t_r \in T$.

□

The second property of min-ω-cuts explains the relevance of min-cut optimality. It states that, for a schedule π, being min-cut optimal implies optimality for the $\preceq$ relation both for π and its reverse $\bar{\pi}$.

Theorem 3. *Let G be a directed acyclic graph with a single source vertex and a single sink vertex, and π be a min-cut optimal schedule of G. Then, π is minimal for $\preceq$ on G and $\bar{\pi}$ is minimal for $\preceq$ on $\bar{G}$.*

PROOF — Consider a directed acyclic graph G with single source and sink vertices, and a min-cut optimal schedule π. Since π is min-cut optimal, π is cut-optimal with some min-ω-cut (S, T). Take any schedule λ of G, and consider the schedule $\gamma = \langle \lambda[S], \lambda[T] \rangle$, which is in compliance with (S, T). We have $\gamma \preceq \lambda$ and $\bar{\gamma} \preceq \bar{\lambda}$ by Theorem 2. Here, we only show $\pi \preceq \gamma$ for G (which

induces $\pi \preceq \lambda$), as with similar arguments one can also show $\bar{\pi} \preceq \bar{\gamma}$ (which induces $\bar{\pi} \preceq \bar{\lambda}$).

By the definition of a min-cut optimal schedule, $\bar{\pi}[S]$ (respectively $\pi[T]$) is minimal for $\preceq$ on S (resp. on T), thus $\bar{\pi}[S] \preceq \bar{\gamma}[S]$ and $\pi[T] \preceq \gamma[T]$. We show that for any vertex $p \in V$, there is a vertex $q \in V$ such that $p \ {}_{\pi}\!\!\rightarrow_{\gamma} \ q$. We consider a vertex $p \in V$ and distinguish two cases depending where p lies:

1. ($p \in S$). Let q be such that $\Sigma(q, \gamma[S]) = \rho(\gamma[S])$. We show $p \ {}_{\pi}\!\!\rightarrow_{\gamma} \ q$ by examining the two conditions.

   (a) (C1: $\Sigma(p, \pi) \leq \Sigma(q, \gamma)$). Since $\bar{\pi}[S] \preceq \bar{\gamma}[S]$, by Corollary 1 we have $\rho(\pi[S]) \leq \rho(\gamma[S])$. Hence,

   $$\Sigma(p, \pi) \leq \rho(\pi[S]) \leq \rho(\gamma[S]) = \Sigma(q, \gamma).$$

   (b) (C2: $\min\{\Sigma(s, \pi) : s \geq_{\pi} p\} \leq \min\{\Sigma(r, \gamma) : r \geq_{\gamma} q\}$). Let t be the sink vertex. Note that $\Sigma(t, \pi) \leq \Sigma(t, \gamma)$ and $t \geq_{\pi} p$. Take any $r \geq_{\gamma} q$ and $r <_{\gamma} t$. For $s^+ = \max^{\pi}\{s \in S\}$, we have $\Sigma(s^+, \pi) = c(S, T) \leq \Sigma(r, \gamma)$. Notice that $s^+ \geq_{\pi} p$, as $p \in S$.

2. ($p \in T$). Let q be such that $p \ {}_{\pi[T]}\!\!\rightarrow_{\gamma[T]} \ q$; such a q exists, because $\pi[T] \preceq \gamma[T]$. We show $p \ {}_{\pi}\!\!\rightarrow_{\gamma} \ q$ by examining the two conditions:

   (a) (C1: $\Sigma(p, \pi) \leq \Sigma(q, \gamma)$). It holds that

   $$\begin{aligned} \Sigma(p, \pi) &= \Sigma(p, \pi[T]) + \Sigma(s^+, \pi[S]) && (9) \\ &\leq \Sigma(q, \gamma[T]) + \Sigma(s^+, \pi[S]) && (10) \\ &= \Sigma(q, \gamma[T]) + \Sigma(s^+, \gamma[S]) = \Sigma(q, \gamma). && (11) \end{aligned}$$

   Equations (9) and (11) come from the fact that π and γ are in compliance with (S, T). Equation (10) is derived from (9) and $p \ {}_{\pi[T]}\!\!\rightarrow_{\gamma[T]} \ q$.

   (b) (C2: $\min\{\Sigma(s, \pi) : s \geq_{\pi} p\} \leq \min\{\Sigma(r, \gamma) : r \geq_{\gamma} q\}$). Take any $r \geq_{\gamma} q$. Then, there exists $s \geq_{\pi} p$ such that $\Sigma(s, \pi[T]) \leq \Sigma(r, \gamma[T])$. Then,

   $$\begin{aligned} \Sigma(s, \pi) &= \Sigma(s, \pi[T]) + \Sigma(s^+, \pi[S]) \\ &\leq \Sigma(r, \gamma[T]) + \Sigma(s^+, \pi[S]) \\ &= \Sigma(r, \gamma[T]) + \Sigma(s^+, \gamma[S]) = \Sigma(r, \gamma). \end{aligned}$$

□

Note that the previous two theorems only apply to directed graphs with a single source and a single sink, and thus in particular to series-parallel graphs. However, it is easy to transform any directed graph so that it obeys these conditions: simply add two zero-weight artificial vertices, one for the source and one for the sink, connect all sources of the original graph to the new source, and connect the new sink to all original sinks.

4.4. Algorithm for general series-parallel graphs

We present here our main contribution, which is a recursive algorithm, called SP-Schedule, that computes a min-cut optimal schedule for a series-parallel graph G in the cumulative weight model. As presented before, this means that the computed schedule is cut-optimal with some min-ω-cut (S, T) and implies that it also minimizes the cutwidth. The algorithm also outputs this (S, T) cut.

This algorithm relies on an algorithm to compute a min-cut optimal schedule of a parallel-chain graph, PC-Schedule which is presented below. This algorithm for parallel-chain graphs is similar to Algorithm 2, but applies to our new cumulative weight model. Likewise, this latter algorithm relies on an algorithm for trees, Tree-Schedule, which is the translation of Liu's algorithm for the cumulative weight model.

Algorithm 3 SP-Schedule(G)

Require: $G = (V, E, \omega)$: series-parallel graph.
Ensure: π: schedule and (S, T): min-ω-cut

▶ Base case
if $|E| = 1$ **then**
  $(S, T) \leftarrow (\{v\}, \{w\})$
  return $[\langle v, w\rangle, (S, T)]$

▶ G is series or parallel comb. of $G_1 = (V_1, E_1, \omega)$ and $G_2 = (V_2, E_2, \omega)$
$[\pi_1, (S_1, T_1)] \leftarrow$ SP-Schedule(G_1)
$[\pi_2, (S_2, T_2)] \leftarrow$ SP-Schedule(G_2)

▶ Series Composition
if $G = \langle G_1, G_2\rangle$ **then**
  $(S, T) \leftarrow \operatorname{argmin} \{(S_1, T_1 \cup V_2), (V_1 \cup S_2, T_2)\}$
  return $[\langle \pi_1, \pi_2\rangle, (S, T)]$

▶ Parallel Composition
if $G = \{G_1, G_2\}$ **then**
  $\widetilde{G}_1 \leftarrow$ Linearize(G_1, π_1)
  $\widetilde{G}_2 \leftarrow$ Linearize(G_2, π_2)
  $(S, T) \leftarrow (S_1 \cup S_2, T_1 \cup T_2)$
  $\pi \leftarrow$ PC-Schedule($\widetilde{G}_1 \cup \widetilde{G}_2, (S, T)$)
  return $[\pi, (S, T)]$

The base case of the algorithm considers a series-parallel graph with a single edge, and outputs the unique schedule along with the unique topological cut.

In the general case, G is a series or parallel composition of two smaller series-parallel graphs G_1 and G_2. We first recursively compute schedules π_1 and π_2 that are cut-optimal with topological cuts (S_1, T_1) and (S_2, T_2) for G_1 and G_2.

If G is a series composition, the final schedule is obtained through a simple concatenation of π_1 and π_2. This schedule of G is cut-optimal with both

topological cuts induced by (S_1, T_1) and (S_2, T_2) on G. Thus, we select the one with lower cutwidth as it is a min-ω-cut of G. Therefore, π is a min-cut-optimal schedule of G.

In case of parallel composition, we transform each subgraphs G_1 and G_2 into a chain using LINEARIZE, based on schedules π_1 and π_2, respectively: $\tilde{G}_1$ (resp. $\tilde{G}_2$) is a chain with the same vertices as G_1 (resp. G_2) such that its unique topological order is π_1 (resp. π_2). The graph obtained by replacing G_1 and G_2 by $\tilde{G}_1$ and $\tilde{G}_2$ is a parallel-chain graph with two chains. We consider the topological-cut (S, T) obtained by unifying the min-ω-cuts of G_1 and G_2: as we will prove later, it is a min-ω-cut of the parallel-chain graph, but also of the original graph. We then call the routine PC-SCHEDULE which finds a schedule π that is cut-optimal with this particular topological cut (S, T).

Algorithm 4 PC-SCHEDULE$(G, (S, T))$

Require: $G = (V, E, \omega)$: parallel-chain directed graph.
Require: (S, T): Min-ω-cut of G.
Ensure: π: Schedule
  $\bar{\sigma} \leftarrow$ TREE-SCHEDULE$(\bar{G}[S])$
  $\tau \leftarrow$ TREE-SCHEDULE$(G[T])$
  return $\langle \sigma, \tau \rangle$

Algorithm 4 presents PC-SCHEDULE, which computes a schedule π that is cut-optimal with the given min-ω-cut (S, T) for a given parallel-chain graph G. The algorithm simply divides the parallel-chain graph into two trees according to (S, T). These trees have a special property: a subtree rooted at every vertex but the root itself has a non-negative total weight. This property follows from the fact that (S, T) is a min-ω-cut of G. The algorithm then schedules the two trees independently using TREE-SCHEDULE routine, which is a translation of Liu's optimal algorithm for trees for the cumulative weight model, and can be found as Algorithm D.1 in Appendix D.

4.5. Correctness of the algorithm

We prove here that SP-SCHEDULE computes a schedule which minimizes the cutwidth (and therefore, thanks to Theorem 1, it also minimizes peak memory). This result is expressed in Theorem 4.

Theorem 4 (Main Theorem). *For any series-parallel graph,* SP-SCHEDULE *computes a schedule π such that the cutwidth of π is optimal.*

Theorem 4 is easily proved below once the following theorem is proved.

Theorem 5. *For any series-parallel graph,* SP-SCHEDULE *computes a schedule π and a min-ω-cut (S, T) such that π is min-cut optimal with (S, T).*

The proof of Theorem 5 is decomposed into several steps, following the different cases in the algorithm. These steps are formalized through the following three theorems.

The first step deals with the series composition. The following theorem states that concatenating two min-cut optimal schedules for the subgraphs leads to a min-cut optimal schedule of their series composition. This case seems straightforward, at least when considering cutwidth minimization: concatenating two schedules with minimal cutwidth on both subgraphs clearly gives a schedule with minimal cutwidth on the whole graph. However, this theorem is stronger and deserves more care as we need to prove min-cut optimality, that is minimality with respect to the $\preceq$ relation.

Theorem 6. *Let G_1 and G_2 be two series-parallel directed graphs. Let π_1 and π_2 each be a min-cut-optimal schedule of G_1 and G_2, respectively. Then, $\pi = \langle \pi_1, \pi_2 \rangle$ is a min-cut-optimal schedule of their series composition $G = \langle G_1, G_2 \rangle$.*

PROOF — The statement follows as a corollary of Lemma B.1 in Appendix B. □

The second step deals with the parallel composition. If we have a min-ω-cut of a series-parallel directed graph and an optimal schedule of each part of this directed graph, then there is an optimal schedule of the whole graph which induces theses two schedules. This theorem is used to compare the schedule returned by SP-SCHEDULE with an optimal one and show that it is indeed optimal itself.

Theorem 7. *Let G_1 and G_2 be two series-parallel directed graphs, and $G = \{G_1, G_2\}$ be their parallel composition. Let (S, T) be a min-ω-cut of G, and π_1 and π_2 be schedules that are cut-optimal with the topological cuts induced by (S, T) on G_1 and G_2, respectively. For each schedule γ in compliance with (S, T) of G, there is a schedule π in compliance with (S, T) of G such that*

(i) *π induces π_1 and π_2,*

(ii) *$\bar{\pi}[S] \preceq \bar{\gamma}[S]$ and $\pi[T] \preceq \gamma[T]$.*

PROOF — See Appendix C. □

The final step is to prove that the PC-SCHEDULE algorithm used to process parallel-chain graphs produces a min-cut optimal schedule compatible with the provided cut.

Theorem 8. *For any parallel-chain graph G and any min-ω-cut (S, T) of G,* PC-SCHEDULE *computes a schedule π that is cut-optimal with (S, T).*

PROOF — See Appendix D. □

We are now ready to prove the main theorem by proving Theorem 5.
PROOF — (Proof of Theorem 5) The proof is by induction on the number of edges of G. In the base case, G has a single edge, that connects the source to the sinks. There is a unique schedule of G, which is also cut-optimal with the unique topological cut.

We now assume that the theorem holds for any series-parallel graph with fewer than k edges, where $k > 1$, and consider a series-parallel graph G with exactly k edges. G is made by the (series or parallel) composition of two series-parallel graphs G_1 and G_2. Both subgraphs have fewer than k edges, thus by induction hypothesis, the recursive calls to SP-Schedule produce two min-cut-optimal schedules π_1 and π_2 together with their respective min-ω-cuts (S_1, T_1) and (S_2, T_2). We now distinguish between the two possible compositions.

Series composition. Theorem 6 shows that the schedule $\pi = \langle \pi_1, \pi_2 \rangle$ computed by SP-Schedule is a min-cut-optimal schedule of G. Besides a min-ω-cut of G can simply be found by selecting one of the min-ω-cut of G_1 and G_2 with minimal cutwidth.

Parallel composition. Now, we consider that $G = \{G_1, G_2\}$. The algorithm first compute the cut (S, T) such that $S = S_1 \cup S_2$ and $T = T_1 \cup T_2$. We first prove that (S, T) is a min-ω-cut of G. For the sake of contradiction assume the contrary. It means that there exists a topological cut (S^*, T^*) such that $c(S^*, T^*) < c(S, T)$. Let (S_1^*, T_1^*) and (S_2^*, T_2^*) be the topological cuts induced by (S^*, T^*) on G_1 and G_2, respectively. Then,

$$\begin{aligned} c(S_1^*, T_1^*) + c(S_2^*, T_2^*) &= c(S^*, T^*) \\ &< c(S, T) \\ &= c(S_1, T_1) + c(S_2, T_2). \end{aligned}$$

This implies $c(S_1^*, T_1^*) < c(S_1, T_1)$ or $c(S_2^*, T_2^*) < c(S_2, T_2)$, which is a contradiction as (S_1, T_1) and (S_2, T_2) both are min-ω-cuts.

In the case of a parallel composition, the algorithm computes the chain graph $\widetilde{G}_1$ (respectively $\widetilde{G}_2$) by sequencing the vertices of G_1 (resp. G_2) in the order of π_1 (resp. π_2) and calls PC-Schedule on the parallel-chain graph $\widetilde{G}$ made by the parallel composition of $\widetilde{G}_1$ and $\widetilde{G}_2$ and the cut (S, T). Thanks to Theorem 8, we know that PC-Schedule outputs a schedule π cut-optimal with (S, T). Thus, for any schedule γ of $\widetilde{G}$ in compliance with (S, T), we have

$$\bar{\pi}[S] \preceq \bar{\gamma}[S], \text{ and } \pi[T] \preceq \gamma[T]. \tag{12}$$

We now show that π is cut-optimal with (S, T) for G as well, that is, the previous inequalities also holds for any schedule γ of G in compliance with (S, T). We consider such a schedule γ. Theorem 7 proves that there exists a schedule π^* that (i) induces π_1 and π_2, (ii) π^* is in compliance with (S, T), and (iii)

$$\bar{\pi^*}[S] \preceq \bar{\gamma}[S], \text{ and } \pi^*[T] \preceq \gamma[T]. \tag{13}$$

As π^* induces π_1 and π_2, π^* is a valid schedule of $\widetilde{G}$, we may thus apply Equation (12) to $\gamma = \pi^*$. Combining it with Equation (13), we get

$$\bar{\pi}[S] \preceq \bar{\pi^*}[S] \preceq \bar{\gamma}[S], \text{ and } \pi[T] \preceq \pi^*[T] \preceq \gamma[T].$$

As $\preceq$ is transitive, we have $\bar{\pi}[S] \preceq \bar{\gamma}[S]$ and $\pi[T] \preceq \gamma[T]$. □

PROOF — Proof of the Main Theorem 4 SP-SCHEDULE computes a schedule π and a min-ω-cut such that π is min-cut optimal with the found cut (see Theorem 5). Theorem 3 implies that π is also minimal for the $\preceq$ relation, which in turn means that π minimizes the cutwidth $\rho(\pi)$ due to Corollary 1. □

Algorithm complexity

The PC-SCHEDULE algorithm calls twice Liu's algorithm for trees, whose worst-case complexity for an input of size n is a $O(n^2)$. Except for the recursive calls, this is the most costly step of the SP-SCHEDULE. Since there are at most $O(n)$ recursive calls, the overall worst-case complexity of SP-SCHEDULE is $O(n^3)$.

5. Conclusion

In this paper, we have proposed an algorithm to schedule a series-parallel task graph with the minimum peak memory. To prove the correctness of the algorithm, we have introduced a new and simpler model, based on a vertex-weighted graph. We have shown how to simulate task graphs with weights on vertices and edges using the new model. The proof of the proposed algorithm is complex, and consists in an extension of Liu's work on trees [18].

The use of the proposed algorithm is limited to task graphs structured as series-parallel computations, which constitutes an important class of scientific computing applications. However, this algorithm may be applied to general graphs, if we first transform them into series-parallel graphs, for example using the SP-ization process [12], which is also suggested in the literature [2]. This will lead to suboptimal schedules, because some fictitious edges are added to a graph when it is turned into a series-parallel graph. Future research directions include looking for optimal schedules for other regular graph structures that appear in scientific computing (such as 2D or 3D meshes) structures, which could be approached using existing work on unweighted 2D grids by Diaz et al. [4, 6].

Acknowledgement

We thank Henri Casanova for his comments on an earlier version of this manuscript. Loris Marchal and Bora Uçar were supported in part by French National Research Agency (ANR) project SOLHAR (ANR-13-MONU-0007).

Appendix

The appendix is composed of four parts. Appendix A contains lemmas used in the proof of Theorem 1 in which we proved that minimizing the cutwidth in the cumulative weight model minimizes the peak memory in the original model. The sections Appendix B, C, and D contain lemmas used in showing the correctness of SP-Schedule algorithm. These three sections give the proof of Theorems 6, 7, and 8, respectively. The last section Appendix D also contains the adaptation of Liu's algorithm to the cumulative weight model.

A. Problem transformation

Lemmas A.1–A.4 are used in the proof of Theorem 1 and assume the same notation. In particular, the peak memory minimization problem is given by a graph $G = (V, E, w_n, w_e)$, and the corresponding cutwidth instance has the bipartite graph $G_B = (V_B, E_B, \omega)$.

Lemma A.1. *Let π_B be a schedule of G_B. There exists a vertex $p \in G$ such that the cutwidth of π_B is reached on p_{start}, i.e., $\rho(\pi_B) = \Sigma(p_{start}, \pi_B)$.*

PROOF — For the sake of contradiction, assume that $\rho(\pi_B) > \Sigma(p_{start}, \pi_B)$ for every $p \in G$. Consider a vertex q of G such that the cutwidth of π_B is reached on q_{stop}, i.e., $\rho(\pi) = \Sigma(q_{stop}, \pi_B)$.

By construction (4), $\omega(v_{stop}) \leq 0$, for any vertex v. We consider p_{start}, the last "start" vertex scheduled before q_{stop} in π_B: $p_{start} = \max^{\pi}(\{v_{start} \leq_{\pi_B} q_{stop}\})$. Nnote that there exists such a p_{start} since q_{start} is scheduled before q_{stop}. Then,

$$\Sigma(q_{stop}, \pi_B) = \Sigma(p_{start}, \pi_B) + \sum_{p_{start} <_{\pi_B} v_{stop} \leq_{\pi_B} q_{stop}} \omega(v_{stop}) \leq \Sigma(p_{start}, \pi_B).$$

Then, we have

$$\rho(\pi_B) = \Sigma(q_{stop}, \pi_B) \leq \Sigma(p_{start}, \pi_B),$$

which contradicts that $\rho(\pi_B) > \Sigma(p_{start}, \pi_B)$. □

Lemma A.2. *Let π be a schedule of G. Then, for any $p \in G$,*

$$\mu(p, \pi) = \omega(p_{start}) + \sum_{v <_{\pi} p} (\omega(v_{start}) + \omega(v_{stop})).$$

PROOF — For this proof, we extend the definition of the edge weights in such a way that $w_e(q, r) = 0$ if there is no edge (q, r) in E. With this extension, we may write

$$\begin{aligned} \omega(v_{start}) &= w_n(v) + \sum_{v \leq_{\pi} r} w_e(v, r), \\ \omega(v_{stop}) &= -w_n(v) - \sum_{q \leq_{\pi} v} w_e(q, v). \end{aligned}$$

Now, consider any vertex $p \in G$. We have

$$\begin{aligned}
\mu(p,\pi) &= w_n(p) + \sum_{q \leq_\pi p \leq_\pi r} w_e(q,r) \\
&= w_n(p) + \sum_{p \leq_\pi r} w_e(p,r) + \sum_{q <_\pi p \leq_\pi r} w_e(q,r) \\
&= \omega(p_{start}) + \sum_{q <_\pi p \leq_\pi r} w_e(q,r) + \sum_{q \leq_\pi r <_\pi p} w_e(q,r) - \sum_{q \leq_\pi r <_\pi p} w_e(q,r) \\
&= \omega(p_{start}) + \sum_{q <_\pi p} \left(\sum_{p \leq_\pi r} w_e(q,r) + \sum_{q \leq_\pi r <_\pi p} w_e(q,r) \right) - \sum_{q \leq_\pi r <_\pi p} w_e(q,r) \\
&= \omega(p_{start}) + \sum_{q <_\pi p} \sum_{q \leq_\pi r} w_e(q,r) - \sum_{r <_\pi p} \sum_{q \leq_\pi r} w_e(q,r) \\
&= \omega(p_{start}) + \sum_{v <_\pi p} (\omega(v_{start}) - w_n(v)) - \sum_{v <_\pi p} (-\omega(v_{stop}) - w_n(v)) \\
&= \omega(p_{start}) + \sum_{v <_\pi p} (\omega(v_{start}) - w_n(v) + \omega(v_{stop}) + w_n(v)) \\
&= \omega(p_{start}) + \sum_{v <_\pi p} (\omega(v_{start}) + \omega(v_{stop})).
\end{aligned}$$

□

Lemma A.3. *Let π_B be a schedule of G_B such that p_{start} and p_{stop} are consecutive in π_B, for each $p \in G$. Consider the schedule π of G such that $p \leq_\pi q$ whenever $p_{stop} \leq_{\pi_B} q_{stop}$. Then, $\mu(\pi) = \rho(\pi_B)$.*

PROOF — As p_{start} and p_{stop} are consecutive in π_B, for each $p \in G$, we have

$$\Sigma(p_{start}, \pi_B) = \omega(p_{start}) + \sum_{v <_\pi p} (\omega(v_{start}) + \omega(v_{stop}))$$

and, due to Lemma A.2, $\Sigma(p_{start}, \pi_B) = \mu(p, \pi)$, for any $p \in G$. We consider the vertex p which reaches the cutwidth of π_B, as defined by Lemma A.1: $\rho(\pi_B) = \Sigma({p_{start}}^*, \pi_B)$. Then,

$$\rho(\pi_B) = \Sigma(p_{start}, \pi_B) = \mu(p, \pi) \leq \mu(\pi). \tag{A.1}$$

Let $q \in G$ be such that $\mu(\pi) = \mu(q, \pi)$. Then,

$$\mu(\pi) = \mu(q, \pi) = \Sigma(q_{start}, \pi_B) \leq \rho(\pi_B). \tag{A.2}$$

By combining Equations A.1 and A.2, we have $\mu(\pi) = \rho(\pi_B)$. □

Lemma A.4. *Let π_B be a schedule of G_B. There exists a schedule π'_B of G_B such that (i) p_{start} and p_{stop} are consecutive in π'_B, for each $p \in G$, and (ii) $\rho(\pi'_B) \leq \rho(\pi_B)$.*

PROOF — Let $p \in G$ be such that p_{start} and p_{stop} are not consecutive in π_B. We construct a schedule π''_B from π_B, where p_{start} is placed right before p_{stop}. We show that π''_B has a cutwidth that is smaller than or equal to that of π_B.

Let $q \in G$ be such that $\rho(\pi''_B) = \Sigma(q_{start}, \pi''_B)$, which exists due to Lemma A.1. We investigate the following two cases of q:

- q_{start} is scheduled before p_{start} in π_B. Then, moving p_{start} forward does not influence the cutwidth.
- q_{start} is scheduled after p_{start} in π_B.

  We first show that $\Sigma(q_{start}, \pi''_B) \leq \Sigma(q_{start}, \pi_B)$ as follows. If $q_{start} >_{\pi_B} p_{stop}$, then $\Sigma(q_{start}, \pi''_B) = \Sigma(q_{start}, \pi_B)$. Otherwise, since $\omega(p_{stop}) \leq 0$, we have $\Sigma(q_{start}, \pi''_B) = \Sigma(q_{start}, \pi_B) + \omega(p_{stop}) \leq \Sigma(q_{start}, \pi_B)$. Thus, we have $\rho(\pi''_B) = \Sigma(q_{start}, \pi''_B) \leq \Sigma(q_{start}, \pi_B) \leq \rho(\pi_B)$.

We repeat this process until p_{start}, p_{stop} vertices are scheduled consecutively, which does not degrade the cutwidth. □

B. Series composition (used for Theorem 6)

Lemma B.1. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two series-parallel directed graphs. Let (S_1, T_1) and (S_2, T_2) be min-ω-cuts, and π_1 and π_2 be schedules cut-optimal with (S_1, T_1) and (S_2, T_2) for G_1 and G_2, respectively. Let $\pi = \langle \pi_1, \pi_2 \rangle$ be the schedule of the series composition $G = \langle G_1, G_2 \rangle$ inducing π_1 and π_2. Then,*

(i) *π is cut-optimal with both $(S_1, T_1 \cup V_2)$ and $(V_1 \cup S_2, T_2)$,*

(ii) *(S, T) is a min-ω-cut of G, where*

$$(S, T) = \mathit{argmin}\ \{(S_1, T_1 \cup V_2), (V_1 \cup S_2, T_2)\}.$$

PROOF — **We first concentrate on item** (i) **of the lemma.** We only show that π is cut-optimal with $(S_1, T_1 \cup G_2)$, as the other can be proven similarly. Let $(S, T) = (S_1, T_1 \cup G_2)$. We need to show $\bar{\pi}[S]$ and $\pi[T]$ both are minimal for $\preceq$. First consider $\bar{\pi}[S]$. Since π_1 is cut-optimal with (S_1, T_1) for G_1, $\bar{\pi}_1[S_1]$ is minimal for $\preceq$ on G_1. Then, $\bar{\pi}[S]$ is minimal for $\preceq$ on G.

Now consider $\tau = \pi[T]$. For any schedule κ of $G[T]$, we need to show $\tau \preceq \kappa$. Take any schedule κ of $G[T]$. We show that there is a vertex $q \in T$ for each vertex $p \in T$ such that $p \ {}_{\tau}\!\!\rightarrow_{\kappa} q$.

Take any $p \in T$. We consider two scenarios of $p \in T_1$ and $p \in G_2$, separately.

a) $\big(p \in T_1\big)$. Let $\tau_1 = \tau[T_1] = \pi_1[T_1]$ and $\kappa_1 = \kappa[T_1]$. Notice that π_1 is min-cut-optimal for G_1. Then, $\pi_1[T_1]$, and equivalently τ_1, is minimal for $\preceq$ by definition. Then, $\tau_1 \preceq \kappa_1$. Then, there is a vertex $q^* \in T_1$ such that $p \ {}_{\tau_1}\!\!\rightarrow_{\kappa_1} q^*$. Note that $q^* \in T_1 \subseteq T$, and we show $p \ {}_{\tau}\!\!\rightarrow_{\kappa} q^*$, as follows.

1) (C1: $\Sigma(p,\tau) \leq \Sigma(q^*,\kappa)$). $\Sigma(p,\tau) = \Sigma(p,\tau_1) \leq \Sigma(q^*,\kappa_1) = \Sigma(q^*,\kappa)$.

2) (C2: $\min\{\Sigma(s,\tau) : s \geq_\tau p\} \leq \min\{\Sigma(r,\kappa) : r \geq_\kappa q^*\}$). We examine a vertex $r \geq_\kappa q^*$, in relation to the vertex $t^o = \max^\kappa T = \max^\tau T$, which is the single sink vertex of G.

   Case 1. $\big(r \in T_1\big)$. Since $p \ {}_{\tau_1}\!\!\rightarrow_{\kappa_1} q^*$, there is a vertex $s \in T_1$ and $s \geq_{\tau_1} p$ such that $\Sigma(s,\tau_1) \leq \Sigma(r,\kappa_1)$. Then, $s \geq_\tau p$ and

$$\Sigma(s,\tau) = \Sigma(s,\tau_1) \leq \Sigma(r,\kappa_1) = \Sigma(r,\kappa).$$

   Case 2. $\big(r \in G_2, r <_\kappa t^o\big)$. Consider $s \in G_2$ such that $\Sigma(s,\pi_2) = c(S_2,T_2)$. Then, $\Sigma(s,\pi_2) \leq \Sigma(r,\kappa_2)$, where $\kappa_2 = \kappa[G_2]$, due to Lemma 6. Then, $s \geq_\tau p$ and

$$\begin{aligned} \Sigma(s,\tau) &= \Sigma(s,\pi_2) + \omega(T_1) \\ &\leq \Sigma(r,\kappa_2) + \omega(T_1) = \Sigma(r,\kappa). \end{aligned}$$

   Case 3. $\big(r = t^o\big)$. Consider $s = t^o$. Then, $s \geq_\tau p$, and

$$\Sigma(s,\tau) = \Sigma(t^o,\tau) = \Sigma(t^o,\kappa) = \Sigma(r,\kappa).$$

b) $\big(p \in G_2\big)$. Let $\tau_2 = \tau[G_2] = \pi_2$ and $\kappa_2 = \kappa[G_2]$. Notice that π_2 is min-cut-optimal for G_2. Then, π_2, and equivalently τ_2, is minimal for $\preceq$ on G_2, due to Theorem 3. Then, $\tau_2 \preceq \kappa_2$. Then, there is a vertex $q^* \in G_2$ such that $p \ {}_{\tau_2}\!\!\rightarrow_{\kappa_2} q^*$. Notice that $\Sigma(v,\tau) = \Sigma(v,\tau_2) + \omega(T_1)$ and $\Sigma(v,\kappa) = \Sigma(v,\kappa) + \omega(T_1)$, for any vertex $v \in G_2$. Since both $p \in G_2$ and $q^* \in G_2$, and $p \ {}_{\tau_2}\!\!\rightarrow_{\kappa_2} q^*$, it holds $p \ {}_{\tau}\!\!\rightarrow_{\kappa} q^*$, and $q^* \in G_2 \subseteq T$.

We now concentrate on item (ii) **of the lemma.** We recall that (S_1,T_1) and (S_2,T_2) are both min-ω-cuts. Now, for the sake of contradiction, suppose that none of $(S_1, T_1 \cup G_2)$ and $(G_1 \cup S_2, T_2)$ is a min-ω-cut of G. Take any min-ω-cut (S^*,T^*) of G. Consider the vertex, say v, that is shared by G_1 and G_2. We analyze the cases of $v \in S^*$ and $v \in T^*$, as follows.

a) $\big(v \in S^*\big)$. Let (S_2^*,T_2^*) be the topological cut induced by (S^*,T^*) on G_2. Then,

$$\begin{aligned} c(S_2^*,T_2^*) &= c(G_1 \cup S_2^*, T_2^*) - \omega(G_1) \\ &= c(S^*,T^*) - \omega(G_1) \\ &< c(G_1 \cup S_2, T_2) - \omega(G_1) = c(S_2,T_2). \end{aligned}$$

This contradicts with the fact that (S_2,T_2) is a min-ω-cut of G_2.

b) $\big(v \in T^*\big)$. Let (S_1^*, T_1^*) be the topological cut induced by (S^*, T^*) on G_1. Then,

$$\begin{aligned} c(S_1^*, T_1^*) &= c(S_1^*, T_1^* \cup G_2) \\ &= c(S^*, T^*) \\ &< c(S_1, T_1 \cup G_2) = c(S_1, T_1). \end{aligned}$$

This contradicts with the fact that (S_1, T_1) is a min-ω-cut of G_1.

□

C. Compatible orders are sufficient (Theorem 7)

In this section, we prove Theorem 7 which allows to transform subgraphs into chains using LINEARIZE before their parallel composition in Algorithm 3. We first define the segmentation of a graph into blocks, which is a coarsening of the graph following a topological order.

Definition C.1 (Segmentation into blocks). *Let G be an directed acyclic graph, and κ be a schedule of G. We say $\mathcal{Q}_\kappa = \{Q_1, Q_2, \ldots\}$ is a* segmentation into blocks *on G for κ, if $\mathcal{Q}_\kappa$ is a partition of the vertices of G, and for any two vertices $p \in Q_j$ and $q \in Q_k$, $p \leq_\kappa q$ implies $j \leq k$. Each $Q_j \in \mathcal{Q}_\kappa$ is called a* block *of $\mathcal{Q}_\kappa$. We define the highest value $\mathrm{H}(Q_j)$ of a block $Q_j \in \mathcal{Q}_\kappa$ as*

$$\mathrm{H}(Q_j) = \max_{q \in Q_j} \Sigma(q, \kappa).$$

Similar to that of Liu [18], we define the hill-valley segmentation of a schedule of a directed acyclic graph, as follows.

Definition C.2 (Hill-valley segmentation). *Let G be a directed acyclic graph, and τ be a schedule of G. The* hill-valley segmentation $\mathcal{P}_\tau = \{P_1, P_2, \ldots\}$ *is a particular block segmentation into blocks on G for τ, which is defined as follows. Let $v_0 = \min^\tau G$ and*

$$h_1 = \max^\tau \{h \in G : \Sigma(h, \tau) = \rho(\tau)\}.$$

Define v_i and h_i, recursively, as

$$v_i = \max^\tau \{\arg\min_{v \geq_\tau h_i} \Sigma(v, \tau)\},$$

and

$$h_i = \max^\tau \{\arg\max_{h >_\tau v_{i-1}} \Sigma(h, \tau)\}.$$

Finally, define $P_1 = \{v : v_0 \leq_\tau v \leq_\tau v_1\}$, and $P_i = \{v : v_{i-1} < v \leq v_i\}$, for $i > 1$. We say that each $P_i \in \mathcal{P}_\tau$ is a segment *of $\mathcal{P}_\tau$. The hill-value H_i and the valley-value V_i of a segment $P_i \in \mathcal{P}_\tau$ are defined as*

$$\mathrm{H}_i = \Sigma(h_i, \tau), \text{ and } \mathrm{V}_i = \Sigma(v_i, \tau).$$

Note that $\Sigma(p, \tau) \leq \mathrm{H}_i$, for any $p \in P_i$.

The following property, similar to that given by Liu [18, Lemma 5.1], of the hill-valley segmentation is clear from the definition.

Lemma C.1. *Let G be a directed acyclic graph, τ be a schedule of G, and $\mathcal{P}_\tau = \{P_1, P_2, \ldots, P_r\}$ be the hill-valley segmentation of G for τ. Then,*

$$\mathrm{H}_1 > \mathrm{H}_2 > \cdots > \mathrm{H}_r \geq \mathrm{V}_r > \cdots > \mathrm{V}_2 > \mathrm{V}_1.$$

Hereafter, we assume that $\mathcal{P}_\tau$ implies a hill-valley segmentation of G for a schedule τ, implicitly. Similarly, $\mathcal{Q}_\kappa$ implies any arbitrary segmentation, unless explicitly stated, of G into blocks for a schedule κ.

Definition C.3 (Segment indicates block). *Let G be a directed acyclic graph, τ and κ be schedules of G. Let $P_i \in \mathcal{P}_\tau$ be a segment and $Q_j \in \mathcal{Q}_\kappa$ be a block. We write $P_i \to Q_j$, and read P_i indicates Q_j, if there is a vertex $q_j^* \in Q_j$ such that $h_i \;{}_\tau\!\!\rightarrow_\kappa q_j^*$.*

We now demonstrate some properties of the hill-valley segmentation. For Propositions C.1–C.4, we assume G is any arbitrary directed acyclic, and τ and κ are schedules of G.

Proposition C.1. *Let $P_i \in \mathcal{P}_\tau$ be a segment and $Q_j \in \mathcal{Q}_\kappa$ be a block. If $P_i \to Q_j$, then*

(i) $\mathrm{H}_i \leq \mathrm{H}(Q_j)$,

(ii) $\mathrm{V}_i \leq \min\{\Sigma(r, \kappa) : r \geq_\kappa q_j^o\}$, *where* $q_j^o = \max^\kappa Q_j$.

PROOF — Assume $P_i \to Q_j$. Then, there is a $q_j^* \in Q_j$ such that $h_i \;{}_\tau\!\!\rightarrow_\kappa q_j^*$, following the definition of dominance given in Section 4.2.1. We prove the two results separately as follows.

(i) By the first condition (C1) of $h_i \;{}_\tau\!\!\rightarrow_\kappa q_j^*$, we conclude

$$\mathrm{H}_i = \Sigma(h_i, \tau) \leq \Sigma(q_j^*, \kappa) \leq \mathrm{H}(Q_j).$$

(ii) By the second condition (C2) of $h_i \;{}_\tau\!\!\rightarrow_\kappa q_j^*$, we have

$$\begin{aligned} \mathrm{V}_i &= \min\{\Sigma(s, \tau) : s \geq_\tau h_i\} \\ &\leq \min\{\Sigma(r, \kappa) : r \geq_\kappa q_j^*\} \leq \min\{\Sigma(r, \kappa) : r \geq_\kappa q_j^o\}. \end{aligned}$$

□

Proposition C.2. *Let $P_i \in \mathcal{P}_\tau$ be a segment and $Q_j \in \mathcal{Q}_\kappa$ be a block. Let $Q_k \in \mathcal{Q}_\kappa$ be another block with $k < j$. If $P_i \to Q_k$ and $P_i \not\to Q_j$, then $\mathrm{H}(Q_j) < \mathrm{H}_i$.*

PROOF — Suppose, for the sake of contradiction, that $P_i \to Q_k$ and $P_i \not\to Q_j$ but $\mathrm{H}(Q_j) \geq \mathrm{H}_i$. Since $P_i \to Q_k$, there is $q_k^* \in Q_k$ such that $h_i \ {}_\tau\!\!\to_\kappa q_k^*$. Then,

$$\min\{\Sigma(s,\tau) : s \geq_\tau h_i\} \leq \min\{\Sigma(r,\kappa) : r \geq_\kappa q_k^*\}.$$

Consider $q_j^* \in Q_j$ such that $\Sigma(q_j^*, \kappa) = \mathrm{H}(Q_j)$. Then, $q_k^* \leq_\kappa q_j^*$, and thus,

$$\min\{\Sigma(s,\tau) : s \geq_\tau h_i\} \leq \min\{\Sigma(r,\kappa) : r \geq_\kappa q_j^*\}. \tag{C.1}$$

Since $\mathrm{H}_i \leq \mathrm{H}(Q_j)$, as we have supposed for the sake of contradiction, we have

$$\Sigma(h_i, \tau) = \mathrm{H}_i \leq \mathrm{H}(Q_j) = \Sigma(q_j^*, \kappa). \tag{C.2}$$

Then, Equations (C.1) and (C.2) together imply $h_i \ {}_\tau\!\!\to_\kappa q_j^*$, which arises a contradiction with $P_i \not\to Q_j$. □

Proposition C.3. *Let $\tau \preceq \kappa$. For any segment $P_i \in \mathcal{P}_\tau$, there is a block $Q_j \in \mathcal{Q}_\kappa$ such that $P_i \to Q_j$.*

PROOF — Take any $P_i \in \mathcal{P}_\tau$. Since $\tau \preceq \kappa$, we have $h_i \ {}_\tau\!\!\to_\kappa q_j^*$, for some $q_j^* \in G$. For $q_j^* \in Q_j$, this implies $P_i \to Q_j$, by Definition C.3 □

Definition C.4 (Monotonic segments-to-blocks function). *A function $g : \mathcal{P}_\tau \to \mathcal{Q}_\kappa$ is called monotonic if*

(i) *$P_i \to g(P_i)$, for any $P_i \in \mathcal{P}_\tau$, and*

(ii) *$Q_j = g(P_i)$ and $Q_k = g(P_{i+1})$ implies $j \leq k$.*

Proposition C.4, together with Proposition C.3, proves that there is a monotonic segments-to-blocks function $g_{\tau\to\kappa}$ whenever $\tau \preceq \kappa$.

Proposition C.4. *Let $\tau \preceq \kappa$. For any consecutive segments $P_i, P_{i+1} \in \mathcal{P}_\tau$ and any block $Q_j \in \mathcal{Q}_\kappa$ with $P_i \to Q_j$, there is a block $Q_k \in \mathcal{Q}_\kappa$ such that $P_{i+1} \to Q_k$ and $j \leq k$.*

PROOF — We prove by contradiction. Suppose the contrary that, $P_{i+1} \not\to Q_k$ for any $k \geq j$. Take any $Q_k \in \mathcal{Q}_\kappa$ such that $P_{i+1} \to Q_k$, which exists due to Preposition C.3. Then we have, $P_i \to Q_j$, $P_{i+1} \not\to Q_j$, $P_{i+1} \to Q_k$, and $k < j$.

(1) $\mathrm{H}_{i+1} < \mathrm{H}_i$, due to Lemma C.1,

(2) $\mathrm{H}_i \leq \mathrm{H}(Q_j)$, due to Proposition C.1,

(3) $\mathrm{H}(Q_j) < \mathrm{H}_{i+1}$, due to Proposition C.2.

Then, we obtain $\mathrm{H}_{i+1} < \mathrm{H}_{i+1}$, which arises a contradiction. □

Hereafter, we focus on a particular kind of directed acyclic graph, so called half-series-parallel. We use half-series-parallel directed graphs and Lemma C.4 as building blocks of the proof of Theorem 7.

Definition C.5 (Half-series-parallel directed graph). *A directed acyclic graph H is called* half-series-parallel*, if there is a series-parallel directed graph G and a min-ω-cut (S,T) of G such that $H = G[T]$.*

Lemma C.2. *Let H be a half-series-parallel directed graph, and t^o be the single sink vertex of H. Let τ be a schedule of H. Then, $\Sigma(p,\tau) \geq 0$, for any $p \in H \setminus \{t^o\}$.*

PROOF — Consider a series-parallel directed graph G, a min-ω-cut (S,T) of G such that $G[T] = H$. Take any $p \in H \setminus \{t^o\}$. For the sake of contradiction, suppose $\Sigma(p,\tau) < 0$. Consider the set $P = \{v \in T : v \leq_\tau p\}$. Then, $(S \cup P, T/P)$ is topological and $\omega(P) = \Sigma(p,\tau) < 0$. Therefore, $c(S \cup P, T/P) = \omega(S) + \omega(P) < \omega(S) = c(S,T)$, which contradicts with the minimality of (S,T). □

Lemma C.3. *Let H_1 and H_2 be two half-series-parallel directed graphs, and H be the half-series-parallel directed graph obtained from H_1 and H_2 by unifying their unique sink vertex. Let λ be a schedule of H and $p^* \in H$. Consider $p_i^* = \max^\lambda\{p \in H_i : p \leq_\lambda p^*\}$, for $i = 1, 2$. If p_1^* and p_2^* both are defined, then*

$$\Sigma(p^*, \lambda) = \Sigma(p_1^*, \lambda[H_1]) + \Sigma(p_2^*, \lambda[H_2]).$$

PROOF — Assume p_1^* and p_2^* are defined. Then,

$$\begin{aligned}
\Sigma(p^*,\lambda) &= \omega(\{p \in H : p \leq_\lambda p^*\}) \\
&= \omega(\{p \in H_1 : p \leq_\lambda p^*\}) + \omega(\{p \in H_2 : p \leq_\lambda p^*\}) \\
&= \omega(\{p \in H_1 : p \leq_{\lambda[H_1]} p_1^*\}) + \omega(\{p \in H_2 : p \leq_{\lambda[H_2]} p_2^*\}) \\
&= \Sigma(p_1^*, \lambda[H_1]) + \Sigma(p_2^*, \lambda[H_2]).
\end{aligned}$$

□

Lemma C.4. *Let H_1 and H_2 be two half-series-parallel directed graphs, and H be the half-series-parallel directed graph obtained from H_1 and H_2 by unifying their unique sink vertex. Let τ be a schedule of H_1 minimal for $\preceq$. For any schedule λ of H, there is a schedule η of H such that* (i) *η induces τ, and* (ii) *$\eta \preceq \lambda$.*

Before proving Lemma C.4, we state and prove Propositions C.5–C.6, which are used in the proof of Lemma C.4. For Propositions C.5-C.6, we use the definitions in Lemma C.4. That is, H_1 and H_2 are half-series-parallel directed graphs, and H is the half-series-parallel directed graph obtained from H_1 and H_2 by unifying their sink vertex. Accordingly, λ is a schedule of H and induces κ on H_1 ($\kappa = \lambda[H_1]$), and $\mathcal{Q}_\kappa$ is the segmentation of H_1 into blocks induced by κ. This particular segmentation is built such that for any $p, q \in H_1$ with $p \leq_\kappa q$, where $p \in Q_j$ and $q \in Q_k$, it holds that $j = k$ if and only if there is no $r \in H_2$ with $p \leq_\lambda r \leq_\lambda q$. We assume τ is a schedule of H_1 such that $\tau \preceq \kappa$, and

$g_{\tau\to\kappa} : \mathcal{P}_\tau \to \mathcal{Q}_\kappa$ is a monotonic segments-to-block function (see Definition C.4 and the discussion after it). We introduce a schedule η which is *built upon* $g_{\tau\to\kappa}$ *over* λ, as follows. We replace in λ each block $Q_j \in \mathcal{Q}$ with the segments of

$$g^{-1}_{\tau\to\kappa}(Q_j) = \{P_i \in \mathcal{P}_\tau : g_{\tau\to\kappa}(P_i) = Q_j\}.$$

By virtue of the monotonic character of $g_{\tau\to\kappa}$, we end up with a schedule η that induces τ by the above-mentioned replacement. Note that η is equivalent to λ on H_2, i.e., $\eta[H_2] = \lambda[H_2]$. Figure C.2 illustrates the construction of η and the notations used in the following propositions. Then, there only remains to show that $\eta \preceq \lambda$ in order to prove Lemma C.4. To do so, we need Propositions C.5–C.7, concerned with the schedule η of H that is built upon $g_{\tau\to\kappa}$ over schedule λ of H.

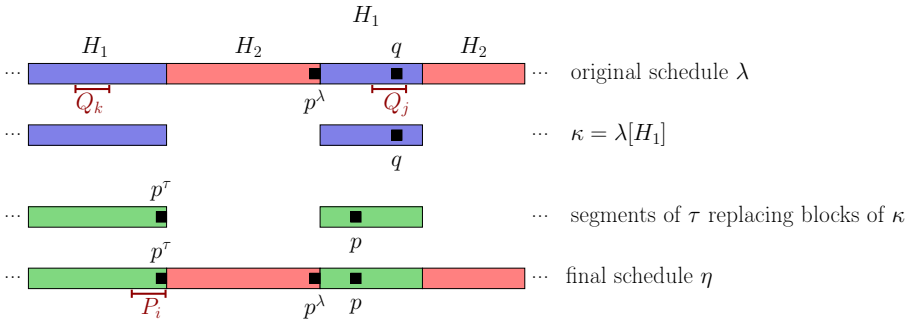


Figure C.1: Illustration of a schedule η built upon $g_{\tau\to\kappa}$ *over* λ, and notations used in Propositions C.5 and C.6.

Proposition C.5. *Let $P_i \in \mathcal{P}_\tau$ be a segment and $Q_j \in \mathcal{Q}_\kappa$ be a block such that $g_{\tau\to\kappa}(P_i) = Q_j$. Recall from above that τ is minimal for $\preceq$ on H_1 and λ is any schedule of H. Let κ be the schedule induced by λ on H_1, and η be the schedule of H built upon $g_{\tau\to\kappa}$ over λ. For any $p \in P_i$ and any $q \in Q_j$, if $\Sigma(p,\tau) \le \Sigma(q,\kappa)$, then $\Sigma(p,\eta) \le \Sigma(q,\lambda)$.*

PROOF — Assume $\Sigma(p,\tau) \le \Sigma(q,\kappa)$. Let $p^\lambda = \max^\lambda\{w \in H_2, w \le_\lambda p\}$. Then, we have two cases to consider.

1. (p^λ is not defined). Then, $\Sigma(p,\eta) = \Sigma(p,\tau) \le \Sigma(q,\kappa) = \Sigma(q,\lambda)$.
2. (p^λ is defined). Then, by the use of Lemma C.3, we have

$$\begin{aligned}\Sigma(p,\eta) &= \Sigma(p,\tau) + \Sigma(p^\lambda, \eta[H_2]) \\ &\le \Sigma(q,\kappa) + \Sigma(p^\lambda, \lambda[H_2]) = \Sigma(q,\lambda).\end{aligned}$$

Note that by construction of η, $p^\lambda = \max^\lambda\{w \in H_2, w \le_\lambda q\}$ which justifies the last equality.

□

Proposition C.6. *Recall that η and λ are schedules of H where η is built upon $g_{\tau\to\kappa}$ over λ using the same τ and κ as before. Let $p \in H_1$ and $p^\lambda = \max^\lambda\{w \in H_2, w \leq_\lambda p\}$. If p^λ is defined, then $\Sigma(p^\lambda, \eta) \leq \Sigma(p, \lambda)$.*

PROOF — Assume p^λ is defined. Let $p^\tau = \max^\eta\{w \in H_1, w \leq_\eta p^\lambda\}$. We consider the following two cases, separately.

1. (p^τ is not defined). Since H_1 is half-series-parallel, $\Sigma(p, \kappa) \geq 0$, as suggested by Lemma C.2. Then, by the use of Lemma C.3, we have

$$\begin{aligned}\Sigma(p^\lambda, \eta) &= \Sigma(p^\lambda, \eta[H_2]) \\ &\leq \Sigma(p^\lambda, \lambda[H_2]) + \Sigma(p, \kappa) = \Sigma(p, \lambda).\end{aligned}$$

2. (p^τ is defined). Let $P_i \in \mathcal{P}_\tau$ such that $p^\tau \in P_i$, and notice that $p^\tau = v_i$. Similarly, let $Q_j \in \mathcal{Q}_\kappa$ such that $p \in Q_j$. Let $Q_k = g_{\tau\to\kappa}(P_i)$, and $q_k^o = \max^\kappa Q_k$. By definition of p^λ and p^τ, we have $k < j$, and thus, $p \geq_\kappa q_k^o$. Note that $P_i \to Q_k$. Then, due to Proposition C.1, we have

$$\Sigma(p^\tau, \tau) = \mathrm{V}_i \leq \min\{\Sigma(r, \kappa) : r \geq_\kappa q_k^o\} \leq \Sigma(p, \kappa).$$

   Then, by Lemma C.3, we have

$$\begin{aligned}\Sigma(p^\lambda, \eta) &= \Sigma(p^\tau, \tau) + \Sigma(p^\lambda, \eta[H_2]) \\ &\leq \Sigma(p, \kappa) + \Sigma(p^\lambda, \lambda[H_2]) = \Sigma(p, \lambda).\end{aligned}$$

□

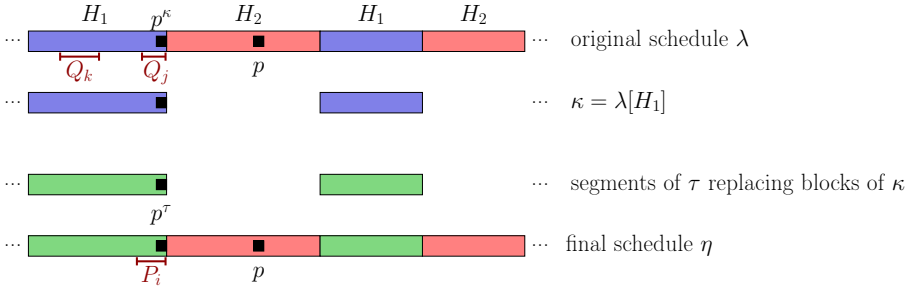


Figure C.2: Illustration of a schedule η built upon $g_{\tau\to\kappa}$ *over* λ, and notations used in Proposition C.7.

Proposition C.7. *Recall that η and λ are schedules of H where η is built upon $g_{\tau\to\kappa}$ over λ using the same τ and κ as before. For any $p \in H_2$, we have $\Sigma(p,\eta) \le \Sigma(p,\lambda)$.*

PROOF — Let $p^\kappa = \max^\lambda\{w \in H_1, w \preceq_\lambda p\}$ and $p^\tau = \max^\eta\{w \in H_1, w \preceq_\eta p\}$. Note that when p^κ is not defined, then p^τ is not defined either. Then, we have the following three cases to consider.

1. (Either p^κ or p^τ is not defined). Then,

$$\Sigma(p,\eta) = \Sigma(p,\eta[H_2]) = \Sigma(p,\lambda[H_2]) = \Sigma(p,\lambda).$$

2. (p^κ is defined, but p^τ is not). Since H_1 is half-series-parallel, $\Sigma(p^\kappa,\kappa) \ge 0$, as Lemma C.2 suggests. Thus, by the use of Lemma C.3, we have

$$\begin{aligned}\Sigma(p,\eta) &= \Sigma(p,\lambda[H_2]) \\ &\le \Sigma(p,\lambda[H_2]) + \Sigma(p^\kappa,\kappa) = \Sigma(p,\kappa).\end{aligned}$$

3. (Both p^κ and p^τ are defined). Let $P_i \in \mathcal{P}_\tau$ such that $p^\tau \in P_i$, and notice that $p^\tau = v_i$. Similarly, let $Q_j \in \mathcal{Q}_\kappa$ such that $p^\kappa \in Q_j$. Let $Q_k = g_{\tau\to\kappa}(P_i)$, and $q_k^o = \max^\kappa Q_k$. By definition of p^κ and p^τ, we have $j \ge k$, and thus, $p^\kappa \succeq_\kappa q_k^o$. Note that $P_i \to Q_k$. Then, due to Proposition C.1, we have

$$\Sigma(p^\tau,\tau) = \mathrm{V}_i \le \min\{\Sigma(r,\kappa) : r \succeq_\kappa q_k^o\} \le \Sigma(p^\kappa,\kappa).$$

Then, by Lemma C.3, we have

$$\begin{aligned}\Sigma(p,\eta) &= \Sigma(p^\tau,\tau) + \Sigma(p,\eta[H_2]) \\ &\le \Sigma(p^\kappa,\kappa) + \Sigma(p,\lambda[H_2]) = \Sigma(p,\lambda).\end{aligned}$$

□

Now, we are ready to prove Lemma C.4.

PROOF — (Proof of Lemma C.4) Recall that τ is a schedule of H_1 which is minimal for $\preceq$, and λ is an arbitrary schedule of H that induces κ on H_1. Since τ is minimal for $\preceq$, $\tau \preceq \kappa$. Then, let $g_{\tau\to\kappa}$ be a monotonic segments-to-blocks function, which exists due to Propositions C.3 and C.4. Let η be a schedule of H built upon $g_{\tau\to\kappa}$ over λ.

With the above construction (i) is proved, as η induces τ in H_1.

We now establish (ii) $\eta \preceq \lambda$. For this, we show that for each $p \in H$, there is $q^* \in H$ such that $p \ {}_\eta\!\to_\lambda q^*$. For any p, either $p \in H_1$ or $p \in H_2$. We explore these two cases separately.

a) $\big(p \in H_1\big)$. Let $p \in P_i$ and $Q_j = g_{\tau\to\kappa}(P_i)$. Since $P_i \to Q_j$, there is a vertex $q_j^* \in Q_j$ such that $p \ {}_\tau\!\to_\kappa q_j^*$. Now, we show that it also holds $p \ {}_\eta\!\to_\lambda q_j^*$.

1) (C1: $\Sigma(p,\eta) \leq \Sigma(q_j^*,\lambda)$). Since $p \ {}_\tau\!\!\rightarrow_\kappa \ q_j^*$, we have $\Sigma(p,\tau) \leq \Sigma(q_j^*,\kappa)$. Then, Proposition C.5 suggests that C1 is met.

2) (C2: $\min\{\Sigma(s,\eta) : s \geq_\eta p\} \leq \min\{\Sigma(r,\lambda) : r \geq_\lambda q_j^*\}$). We show that for each $r \geq_\lambda q_j^*$, there is a vertex $s \geq_\eta p$ such that $\Sigma(s,\eta) \leq \Sigma(r,\lambda)$. We examine three cases: $r \in Q_j$, or $r \in H_1 \setminus Q_j$, or $r \in H_2$.

   Case 1. $\big(r \in Q_j\big)$. Since $p \ {}_\tau\!\!\rightarrow_\kappa \ q_j^*$, there is a vertex $s \geq_\tau p$ such that $\Sigma(s,\tau) \leq \Sigma(r,\kappa)$. We consider two scenarios of $s \in P_i$ and $s \notin P_i$.

   i) $(s \in P_i)$. Then, Proposition C.5 suggests

   $$\Sigma(s,\eta) \leq \Sigma(r,\lambda),$$

   and notice that $s \geq_\eta p$, as $s \geq_\tau p$.

   ii) $(s \notin P_i)$. Then, $s \geq_\tau v_i \geq_\tau p$. Since $\Sigma(v_i,\tau) \leq \Sigma(s,\tau)$, by the definition of v_i, we have $\Sigma(v_i,\tau) \leq \Sigma(r,\kappa)$. Then, Proposition C.5 suggests

   $$\Sigma(v_i,\eta) \leq \Sigma(r,\lambda),$$

   and notice that $v_i \geq_\eta p$, as $v_i \geq_\tau p$.

   Case 2. $\big(r \in H_1 \setminus Q_j\big)$. Let $r \in Q_k$ and $r^\lambda = \max^\lambda\{w \in H_2, w \leq_\lambda r\}$. Since $r \geq_\kappa q_j^*$ and $r \notin Q_j$, it holds $k > j$, and thus, r^λ is defined and $r^\lambda \geq_\lambda q_j^*$. Then, Proposition C.6 suggests

   $$\Sigma(r^\lambda,\eta) \leq \Sigma(r,\lambda),$$

   and notice that $r^\lambda \geq_\eta p$, as $r^\lambda \geq_\lambda q_j^*$.

   Case 3. $\big(r \in H_2\big)$. Then, Proposition C.7 suggests

   $$\Sigma(r,\eta) \leq \Sigma(r,\lambda),$$

   and notice that $r \geq_\eta p$, as $r \geq_\lambda q_j^*$.

b) $\big(p \in H_2\big)$. We show $p \ {}_\eta\!\!\rightarrow_\lambda \ p$. The first condition (C1) directly holds due to Proposition C.7. Considering the second condition (C2), we show, for any $r \geq_\lambda p$, there is $s \geq_\eta p$, such that $\Sigma(s,\eta) \leq \Sigma(r,\lambda)$. We explore the two cases of $r \in H_1$ and $r \in H_2$, separately.

   Case 1. $\big(r \in H_1\big)$. Let $r^\lambda = \max^\lambda\{w \in H_2, w \leq_\lambda r\}$. Since $r \geq_\lambda p$ and $p \in H_2$, r^λ is defined and $r^\lambda \geq_\lambda p$. Due to Proposition C.6, we have

   $$\Sigma(r^\lambda,\eta) \leq \Sigma(r,\lambda),$$

   and notice that $r^\lambda \geq_\eta p$, as $r^\lambda \geq_\lambda p$.

   Case 2. $\big(r \in H_2\big)$. Proposition C.7 suggests

   $$\Sigma(r,\eta) \leq \Sigma(r,\lambda),$$

   and notice that $r \geq_\eta p$, as $r \geq_\lambda p$.

□

We first recall Theorem 7, and show the proof as a consequence of Lemma C.4.

Theorem C.1 (Restatement of Theorem 7). *Let G_1 and G_2 be two series-parallel directed graphs, and $G = \{G_1, G_2\}$ be their parallel composition. Let (S,T) be a min-ω-cut of G, and π_1 and π_2 be schedules that are cut-optimal with the topological cuts induced by (S,T) on G_1 and G_2, respectively. For each schedule γ in compliance with (S,T) of G, there is a schedule π in compliance with (S,T) of G such that*

(i) *π induces π_1 and π_2,*

(ii) *$\bar{\pi}[S] \preceq \bar{\gamma}[S]$ and $\pi[T] \preceq \gamma[T]$.*

PROOF — (Proof of Theorem 7) Take any schedule γ in compliance with (S,T) of G. Let (S_1, T_1) be topological cut induced by (S,T) on G_1. Consider $\lambda^t = \gamma[T]$ and $\tau_1^t = \pi_1[T_1]$. Then, τ_1^t is minimal for $\preceq$, since π_1 is cut-optimal with (S_1, T_1). Then, we have a schedule η^t of $G[T]$ inducing τ_1^t such that $\eta^t \preceq \lambda^t$, as suggested by Lemma C.4. Now, consider $\bar{\lambda}^s = \bar{\gamma}[S]$ and $\bar{\tau}_1^s = \bar{\pi}_1[S_1]$. Then, $\bar{\tau}_1^s$ is minimal for $\preceq$, since π_1 is cut-optimal with (S_1, T_1). Then, we have a schedule $\bar{\eta}^s$ of $G[S]$ inducing $\bar{\tau}_1^s$ such that $\bar{\eta}^s \preceq \bar{\lambda}^s$, as suggested by Lemma C.4. We consider $\dot{\pi} = \langle \eta^s, \eta^t \rangle$. We notice that (i) $\dot{\pi}$ induces π_1 and $\gamma[G_2]$ on G_1 and G_2, respectively. It also holds that (ii) $\dot{\bar{\pi}}[S] \preceq \bar{\gamma}[S]$ and $\dot{\pi}[T] \preceq \gamma[T]$.

Now, apply the procedure given above on $\dot{\pi}$ (instead of γ) for π_2 (instead of π_1. Then, we obtain $\pi = \langle \dot{\eta}^s, \dot{\eta}^t \rangle$ that (i) induces $\dot{\pi}[G_1]$ and π_2 on G_1 and G_2, respectively. It also holds that (ii) $\bar{\pi}[S] \preceq \dot{\bar{\pi}}[S]$ and $\pi[T] \preceq \dot{\pi}[T]$.

As a result, (i) π induces π_1 and π_2. Secondly, since $\preceq$ is transitive, it holds that (ii) $\bar{\pi}[S] \preceq \bar{\gamma}[S]$ and $\pi[T] \preceq \gamma[T]$. □

D. PC-Schedule is min-cut-optimal (Theorem 8)

PC-Schedule, presented in Algorithm 4, relies on Tree-Schedule to compute schedules for trees that are minimal for $\preceq$. This latter algorithm, which we describe below, in turn relies on Liu's algorithm for trees. However, Liu's algorithm uses another model, namely the generalized pebble game: each node is provided with a number of pebbles which should be use to pebble it. As usual in pebble games, the objective is to pebble the tree up to the root using a minimal number of pebbles. In order to prove that his algorithm is optimal, Liu's relies on the notion of *Pcost* sequences. In the following, we prove that a schedule minimal for *Pcost* sequences is also minimal for our relation $\preceq$. The proof of Theorem D.1 thus largely relies on notations and concepts borrowed from [18]. We first introduce a restriction on the trees on which Tree-Schedule may be applied.

Definition D.1 (Liu-compatible tree). *Let T be a directed in-tree and r be its root vertex. We say T is* Liu-compatible *if $\omega(T[v]) \geq 0$ for each vertex $v \in T - \{r\}$, where $T[v]$ is the subtree rooted at v.*

Algorithm D.1 TREE-SCHEDULE$(G, (S, T))$

Require: $G = (V, E, \omega)$: Liu-compatible vertex-weighted tree with root r
Ensure: π: A minimal schedule for $\preceq$
  for $v \in V$ **do**
    if $v = r$ **then**
      $\tau(v) \leftarrow \omega(T - \{r\})$
    else
      $\tau(v) \leftarrow \omega(T - \{r\}[v])$
  Call Algorithm 4.1 in [18] ("Pebble-Ordering") on T, with "Combine" procedure from Algorithm 6.1 (ibid.) to compute schedule π
  return π

Theorem D.1. *For any Liu-compatible tree T, Algorithm D.1 computes a schedule which is minimal for $\preceq$.*

PROOF — For this proof, we use the notations given in [18], and we assume that the reader is familiar with its results. In this article, Liu considers for any vertex v of a tree a non-negative value $\tau(v)$ which represents the number of pebbles required to satisfy this node. We consider here the following pebbling function as defined in Algorithm D.1, namely:

$$\tau(v) = \begin{cases} \omega(T[v]) & v \in T - \{r\} \\ \omega(T - \{r\}) & v = r. \end{cases} \tag{D.1}$$

Notice that $\tau(v) \geq 0$ for each $v \in T$, as T is Liu-compatible. Let PARENT(v) represent the parent vertex of v, for a vertex $v \in T - \{r\}$. As in [18], we consider for any schedule π of T the value $peb_\pi(v)$ which represents the "total number of pebbles used during the pebbling of the vertex v" while following schedule π.

We first prove that with the previous pebbling function, $peb_\pi(v) = \Sigma(v, \pi)$, for each $v \in T - \{r\}$. Take any $v \in T - \{r\}$. Let $F_v = \{r \in T : r \leq_\pi v <_\pi \text{PARENT(r)}\}$, that is, the set populated by the root vertices of the pruned forest. Then,

$$peb_\pi(v) = \sum_{r \in F_v} \tau(r) = \sum_{r \in F_v} \omega(T[r]) = \omega(\{u \leq_\pi v\}) = \Sigma(v, \pi).$$

We then consider the $Pcost(\pi)$, the cost sequence of a schedule π, as defined in [18]. We prove that if $Pcost(\pi) \prec Pcost(\gamma)$ then $\pi \preceq \gamma$, for two schedules π and γ of T.

Assume $Pcost(\pi) \prec Pcost(\gamma)$. Now, for each vertex $p \in T$, we need to exhibit a vertex $q^* \in T$ such that $p \ {}_\pi\!\!\rightarrow_\gamma q^*$. Since $r = \max^\pi T = \max^\gamma T$, we have

$r \ {}_{\pi}\!\!\rightarrow_{\gamma} r$. Now, take any vertex $p \in T - \{r\}$. Let p reside in the i^{th} hill-valley segment with respect to π. By definition of $\prec$ in [18, Section 5.2], there exists j such that $\tilde{H}_i \leq H_j$, and $\tilde{V}_i \leq V_j$, where $(\tilde{H}_i, \tilde{V}_i)$ and (H_j, V_j) are the hill-valley values for i^{th} and j^{th} segments with respect to schedules π and γ, respectively. Consider a vertex $q^* \in T - \{r\}$ so that q^* resides in the j^{th} hill-valley segment with respect to γ, and $peb_\gamma(q^*) = H_j$, where such q^* exists due to Equation D.1. Now, we show $p \ {}_{\pi}\!\!\rightarrow_{\gamma} q^*$ as follows.

1) (C1: $\Sigma(p,\pi) \leq \Sigma(q^*,\gamma)$). Recall that $peb_\pi(p) = \Sigma(p,\pi)$ and $peb_\gamma(q^*) = \Sigma(q^*,\gamma)$. Then, we have:

$$\Sigma(p,\pi) = peb_\pi(p) \leq \tilde{H}_i \leq H_j = peb_\gamma(q^*) = \Sigma(q^*,\gamma).$$

2) (C2: $\min\{\Sigma(s,\pi) : s \geq_\pi p\} \leq \min\{\Sigma(r,\gamma) : r \geq_\gamma q^*\}$). Recall that $peb_\pi(v) = \Sigma(v,\pi)$ and $peb_\pi(v) = \Sigma(v,\gamma)$, for each $v \in T - \{r\}$. By definition of $\tilde{V}_i$ and V_j,

$$\min\{\Sigma(s,\pi) : s \geq_\pi p\} = \tilde{V}_i \leq V_j = \min\{\Sigma(r,\gamma) : r \geq_\gamma q^*\}.$$

Thanks to [18, Theorem 6.4], we known that Algorithm 4.1 in [18] with Combine procedure 6.1 (ibid.) computes a schedule π of T so that $Pcost(\pi) \prec Pcost(\gamma)$, for any schedule γ of T. As a corollary, π is minimal for $\preceq$ on T. □

Theorem D.2 (Restatement of Theorem 8). *For any parallel-chain graph G and any min-ω-cut (S,T) of G,* PC-SCHEDULE *computes a schedule π that is cut-optimal with (S,T).*

PROOF — (Proof of Theorem 8) Let G be a parallel-chain directed graph and (S,T) be a min-ω-cut of G. We first check that both $\bar{G}[S]$ and $G[T]$ are Liu-compatible trees.

We only show that $G[T]$ is Liu-compatible, as the other can be proven similarly. For the sake of contradiction, suppose that $G[T]$ is not Liu-compatible. Then, there is a vertex, say v, such that $\omega(T[v]) < 0$. Consider the topological cut (S',T'), where $S' = S \cup T[v]$ and $T' = G - S'$. Then, $c(S',T') = \omega(S') = \omega(S) + \omega(T[v]) < \omega(S) = c(S,T)$. This contradicts the fact that (S,T) is a min-ω-cut of G.

The first steps of PC-SCHEDULE as described in Algorithm 4 is to compute schedules $\bar{\sigma}$ and τ for $\bar{G}[S]$ and $G[T]$ using TREE-SCHEDULE. Thanks to Theorem D.1, we know that these schedules are minimal for $\preceq$.

Then, by definition, $\pi = \langle \sigma, \tau \rangle$ is cut-optimal with (S,T), as π is in compliance with (S,T), $\bar{\sigma}$ is minimal for $\preceq$ on $\bar{G}[S]$, and τ is minimal for $\preceq$ on $\bar{G}[S]$. □

References

[1] S. Arora, S. Rao, U. Vazirani, Geometry, flows, and graph-partitioning algorithms, Communications of the ACM 51 (10) (2008) 96–105.

[2] G. Cordasco, R. D. Chiara, A. L. Rosenberg, Assessing the computational benefits of area-oriented DAG-scheduling, in: Euro-Par 2011 Parallel Processing - 17th International Conference, Springer, 2011, pp. 180–192.

[3] G. Cordasco, A. L. Rosenberg, On scheduling series-parallel DAGs to maximize area, Int. J. Found. Comput. Sci. 25 (5) (2014) 597–622.

[4] J. Díaz, M. D. Penrose, J. Petit, M. J. Serna, Layout problems on lattice graphs, in: Computing and Combinatorics, 5th Annual International Conference, COCOON '99, Tokyo, Japan, July 26-28, 1999, Proceedings, 1999, pp. 103–112.

[5] J. Díaz, J. Petit, M. Serna, A survey of graph layout problems, ACM Computing Surveys 34 (3) (2002) 313–356.

[6] J. Díaz, J. Petit, M. J. Serna, A survey of graph layout problems, ACM Comput. Surv. 34 (3) (2002) 313–356.

[7] D. Eppstein, Parallel recognition of series-parallel graphs, Information and Computation 98 (1) (1992) 41 – 55.

[8] L. Eyraud-Dubois, L. Marchal, O. Sinnen, F. Vivien, Parallel scheduling of task trees with limited memory, ACM Transactions on Parallel Computing 2 (2) (2015) 13.

[9] L. Finta, Z. Liu, I. Mills, E. Bampis, Scheduling UET-UCT series-parallel graphs on two processors, Theoretical Computer Science 162 (2) (1996) 323–340.

[10] F. Gavril, Some NP-complete problems on graphs, in: 11th Conference on Information Sciences and Systems, The John Hopkins University, Baltimore, Maryland, 1977, pp. 91–95.

[11] J. R. Gilbert, T. Lengauer, R. E. Tarjan, The pebbling problem is complete in polynomial space, SIAM J. Comput. 9 (3) (1980) 513–524.

[12] A. González-Escribano, A. J. C. van Gemund, V. Cardeñoso-Payo, Mapping unstructured applications into nested parallelism, in: High Performance Computing for Computational Science - VECPAR, Springer, 2002, pp. 407–420.

[13] M. Jacquelin, L. Marchal, Y. Robert, B. Uçar, On optimal tree traversals for sparse matrix factorization, in: IPDPS'11, IEEE Computer Society, 2011, pp. 556–567.

[14] C.-C. Lam, T. Rauber, G. Baumgartner, D. Cociorva, P. Sadayappan, Memory-optimal evaluation of expression trees involving large objects, Computer Languages, Systems & Structures 37 (2) (2011) 63–75.

[15] T. Leighton, S. Rao, Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms, J. ACM 46 (6) (1999) 787–832.

[16] J. W. H. Liu, On the storage requirement in the out-of-core multifrontal method for sparse factorization, ACM Trans. Math. Software 12 (3) (1986) 249–264.

[17] J. W. H. Liu, An adaptive general sparse out-of-core Cholesky factorization scheme, SIAM Journal on Scientific and Statistical Computing 8 (4) (1987) 585–599.

[18] J. W. H. Liu, An application of generalized tree pebbling to sparse matrix factorization, SIAM J. Algebraic Discrete Methods 8 (3) (1987) 375–395.

[19] B. Monien, I. H. Sudborough, Min cut is NP-complete for edge weighted trees, Theoretical Computer Science 58 (1) (1988) 209–229.

[20] C. L. Monma, J. B. Sidney, Sequencing with series-parallel precedence constraints, Mathematics of Operations Research 4 (3) (1979) 215–224.

[21] J. Petit, Addenda to the survey of layout problems, Bulletin of the EATCS 105 (2011) 177–201.

[22] Y. Robert, Task graph scheduling, in: D. Padua (ed.), Encyclopedia of Parallel Computing, Springer US, Boston, MA, 2011, pp. 2013–2025.

[23] R. Sethi, Complete register allocation problems, in: STOC'73, ACM Press, 1973, pp. 182–195.

[24] R. Sethi, J. Ullman, The generation of optimal code for arithmetic expressions, J. ACM 17 (4) (1970) 715–728.

[25] D. M. Thilikos, M. Serna, H. L. Bodlaender, Cutwidth I: A linear time fixed parameter algorithm, J. Algorithms 56 (1) (2005) 1–24.

[26] D. M. Thilikos, M. J. Serna, H. L. Bodlaender, A polynomial time algorithm for the cutwidth of bounded degree graphs with small treewidth, in: F. M. auf der Heide (ed.), ESA 2001: 9th Annual European Symposium on Algorithms Århus, Denmark, August 28–31, 2001 Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 380–390.

[27] J. Valdes, R. E. Tarjan, E. L. Lawler, The recognition of series parallel digraphs, SIAM J. Comput. 11 (2) (1982) 298–313.

[28] Y. Wu, P. Austrin, T. Pitassi, D. Liu, Inapproximability of treewidth and related problems, J. Artif. Intell. Res. (JAIR) 49 (2014) 569–600.

[29] M. Yannakakis, A polynomial algorithm for the min-cut linear arrangement of trees, J. ACM 32 (4) (1985) 950–988.